CPSC-406 Report

Ethan Tapia
Chapman University

March 2, 2025

Abstract

TBW

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** 7

1 Introduction

2 Week by Week

2.1 Week 1

Lecture Summary

A finite automaton consists of a finite set of **states** (Q), an **alphabet** (Σ), a **transition function** (δ), a **starting state** (q_0), and a set of **accepting states** (F).

It can be formally represented as:

$$M = (Q, \Sigma, \delta, q_0, F)$$

where:

- Q is the set of states,
- Σ is the input alphabet,
- $\delta : Q \times \Sigma \to Q$ is the transition function,
- $q_0 \in Q$ is the initial state,
- $F \subseteq Q$ is the set of accepting states.

2.2 Week 2

Homework 1

Problem 1: Characterizing Accepted Sequences

The given problem involves designing a finite automaton that accepts sequences of 5 and 10-cent inputs summing to 25 cents.

Solution: We define the equation:

$$5a + 10b = 25 \tag{1}$$

where a is the number of 5-cent inputs and b is the number of 10-cent inputs. Solving for valid pairs:

- $(a = 5, b = 0) \Rightarrow$ Sequence: $5, 5, 5, 5, 5$
- $(a = 3, b = 1) \Rightarrow$ Sequence: $5, 5, 5, 10$
- $(a = 1, b = 2) \Rightarrow$ Sequence: $5, 10, 10$

These sequences are precisely those accepted by the automaton. The machine accepts a sequence if the total sum equals 25 cents.

Problem 2: Defining Valid Variable Names

A valid variable name must begin with a letter (ℓ) and be followed by any number of letters or digits (d).

Regular Expression:

$$\ell(\ell|d)^* \tag{2}$$

Solution: *Finite Automaton:* - States: q_0 (initial), q_1 (accepting). - Transitions: - $q_0 \to q_1$ on input ℓ - $q_1 \to q_1$ on input ℓ or d

Problem 3: Classification of Words in L_1, L_2, L_3

The given languages are defined as follows:

- $L_1 = \{x0y \mid x, y \in \Sigma^*\}$: The set of words that contain at least one '0'.
- $L_2 = \{w \mid |w| = 2^n \text{ for some } n \in \mathbb{N}\}$: The set of words whose length is a power of 2.
- $L_3 = \{w \mid |w|_0 = |w|_1\}$: The set of words where the number of 0s equals the number of 1s.

Solution: We analyze each word based on these conditions:

	L_1	L_2	L_3
$w_1 = 10011$	✓		
$w_2 = 100$	✓		
$w_3 = 10100100$	✓	✓	
$w_4 = 1010011100$	✓		✓
$w_5 = 11110000$	✓	✓	✓

Table 1: Classification of words into L_1, L_2, L_3

Problem 4: DFA Analysis

Given the DFA with states q_0 (start), q_2, and q_1 (accepting), we determine which words end in the accepting state q_1.

Transitions:

$$\begin{aligned} \delta(q_0, 1) &= q_0, & \delta(q_0, 0) &= q_2 \\ \delta(q_2, 0) &= q_2, & \delta(q_2, 1) &= q_1 \\ \delta(q_1, 0) &= q_1, & \delta(q_1, 1) &= q_1 \end{aligned}$$

Checking Words:

- $w_1 = 0010$: $q_0 \to q_2 \to q_2 \to q_1 \to q_1$ ✓(Accepted)
- $w_2 = 1101$: $q_0 \to q_0 \to q_0 \to q_2 \to q_1$ ✓(Accepted)
- $w_3 = 1100$: $q_0 \to q_0 \to q_0 \to q_2 \to q_2$ (Not Accepted)

Solution:

$w_1 = 0010 \to$ ✓*Accepted*
w$_2$ = 1101 $\to$ ✓*Accepted*
w$_3$ = 1100 $\to$ *Rejected*

This confirms that w_1 and w_2 end in the accepting state, while w_3 does not.

Chapter 2.1 Report:

Chapter 2.1 discusses the use of finite automata in modeling real-world protocols, particularly in the context of electronic money transactions. The section introduces a three-party system involving a customer, a store, and a bank. The goal is to ensure that digital money is not duplicated or reused fraudulently.

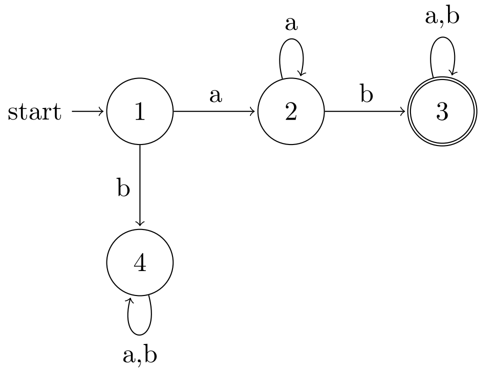
The protocol consists of five primary actions: pay, cancel, ship, redeem, and transfer. Each party's behavior is modeled using finite automata to track transaction states. The section highlights how such models can reveal vulnerabilities—such as a store shipping goods before verifying payment—showcasing the importance of automata in validating protocol security.

Finite automata prove to be useful for detecting logical flaws in transaction systems, ensuring valid sequences of operations. The chapter serves as an introduction to the application of formal computational models in the security and validation of protocols.

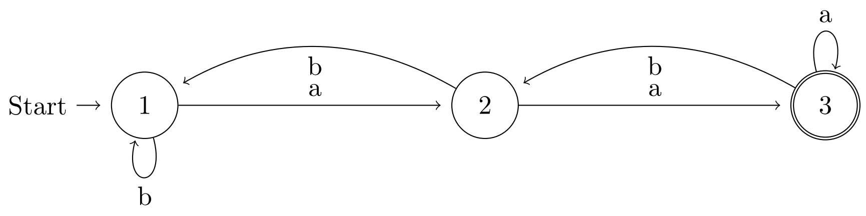
2.3 Week 3

Lecture Summary
TBW

Homework 2
Exercise 2: Implementing DFA Runs



DFA A_1



DFA A_2

Words accepted or refused by $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively

w	Accepted A_1	Accepted A_2
aaa	×	✓
aab	✓	×
aba	×	×
abb	×	×
baa	×	✓
bab	×	×
bba	×	×
bbb	×	×

The table above summarizes the words accepted or rejected by DFA A_1 and DFA A_2. To implement these automata *programmatically*, we define the DFA class in `dfa.py`, which allows us to process input words according to their respective state transition diagrams.

DFA Implementation in `dfa.py`
This introduction describes the design of the `dfa.py`, consisting of:

- Q - a finite set of states.
- Σ - an input alphabet.
- $\delta : Q \times \Sigma \to Q$ - a transition function.
- $q_0 \in Q$ an initial state.
- $F \subseteq Q$ - a set of final accepting states.

The `DFA` class constructor takes these five elements (`Q`, `Sigma`, `delta`, `q0`, and `F`), using the method:

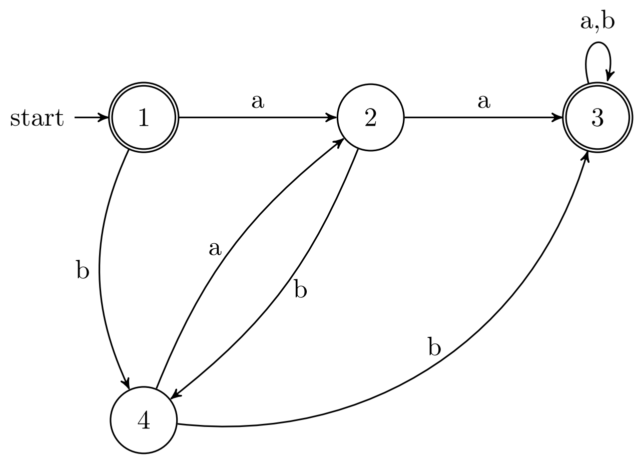
- `run(w)`: Runs the DFA on input string `w` and determines whether or not `w` is accepted based on the state it finishes at.

Implementation In the following code snippet, the `run` method processes the symbols of the input `w` sequentially, looking up the next state based on the current state and the input symbol. If an invalid transition is encountered, the method immediately returns `False`. Otherwise, if the DFA ends in a state that is a member of `F`, `True` is returned (meaning `w` is accepted); if it ends in some other state, `False` is returned.

```
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) :
        # todo
        # start at initial state
        current_state = self.q0
        for symbol in w:
            if (current_state, symbol) in self.delta:
                current_state = self.delta[(current_state, symbol)]
            else:
                # invalid transition (dead state)
                return False
            # accept if in final state
        return current_state in self.F
```

Exercise 4: A new automaton from an old one

Below is DFA A_0 which accepts exactly the words that A refuses and vice versa.



In A_0, nodes 1 and 3 are the accepting final states, while nodes 2 and 4 are normal states.

Exercise 2.2.4: DFAs over $\{0,1\}$

(a) The set of all strings ending in 00

DFA Description:

$$\begin{aligned}&Q = \{\,q_0,\, q_1,\, q_2\},\\ &\Sigma = \{0,1\},\\ &\ \delta \text{ is given by the table below,}\\ &q_0 \text{ is the start state,}\\ &F = \{\,q_2\}.\end{aligned}$$

Transition Table:

δ	0	1
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_0

Explanation:

- q_0 - we have not yet seen a trailing zero
- q_1 - the string currently ends in exactly one zero
- q_2 (accepting) - the string ends in at least two consecutive zeros

(b) The set of all strings with three consecutive 0s

DFA Description:

$$\begin{aligned}&Q = \{\,q_0,\, q_1,\, q_2,\, q_3\},\\ &\Sigma = \{0,1\},\\ &\ \delta \text{ is given by the table below,}\\ &q_0 \text{ is the start state,}\\ &F = \{\,q_3\}.\end{aligned}$$

Transition Table:

δ	0	1
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_3	q_0
q_3	q_3	q_3

Explanation:

- q_0 - we have seen 0 consecutive zeros so far
- q_1 - we have seen exactly 1 consecutive zero
- q_2 - we have seen exactly 2 consecutive zeros
- q_3 (accepting) - we have seen at least 3 consecutive zeros

(c) The set of all strings with `011` as a substring

DFA Description:

$$\begin{aligned}&Q = \{\, q_0,\, q_1,\, q_2,\, q_3\},\\ &\Sigma = \{0,1\},\\ &\ \delta \text{ is given by the table below,}\\ &q_0 \text{ is the start state,}\\ &F = \{\, q_3\}.\end{aligned}$$

Transition Table:

δ	0	1
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_3	q_3

Explanation:

- q_0 - no partial match yet
- q_1 - we matched a single `0`
- q_2 - we matched `01`
- q_3 (accepting) - we found `011` somewhere in the string

2.4 Week 4

Lecture Summary
We explore the concept of *product automata* and how they can be used to model operations on languages accepted by two deterministic finite automata (DFAs). In particular, we discuss how to construct an automaton that accepts the intersection (or union) of the languages recognized by two given automata.

Homework 3

Problem 1: Extended transition function
Consider two DFAs:

$$\mathcal{A}^{(1)} = \big(Q^{(1)}, \Sigma, \delta^{(1)}, q_0^{(1)}, F^{(1)}\big) \quad \text{and} \quad \mathcal{A}^{(2)} = \big(Q^{(2)}, \Sigma, \delta^{(2)}, q_0^{(2)}, F^{(2)}\big),$$

over the alphabet $\Sigma = \{a, b\}$.

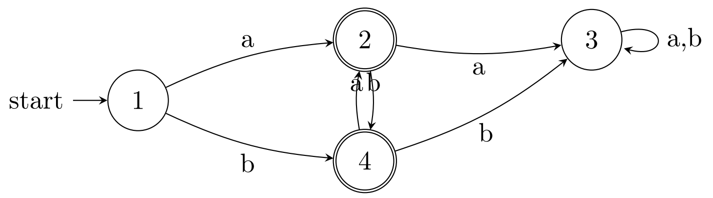


Diagram for $A^{(1)}$:

- State 1 is the initial state (arrow from the left).
- From state 1: reading 'a' goes to 2; reading 'b' goes to 4.

- From state 2: reading 'a' goes to 3; reading 'b' goes to 4.
- From state 4: reading 'a' goes to 2; reading 'b' goes to 3.
- From state 3: reading either 'a' or 'b' loops on 3.
- States 2 and 4 are accepting, while state 3 is a non-accepting sink.

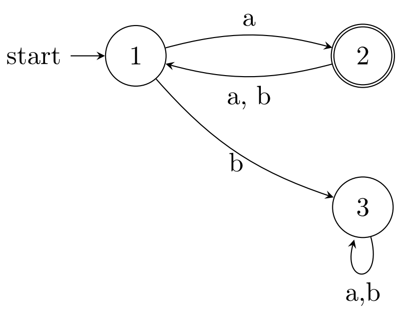


Diagram for $A^{(2)}$:

- State 1 is the initial state (arrow from the left).
- From state 1: reading 'a' goes to 2; reading 'b' goes to 3.
- From state 2: reading 'a' or 'b' goes back to 1.
- State 3 loops on both 'a' and 'b'.
- State 2 is the only accepting state.

a. Descriptions of Accepted Languages

- $\mathcal{A}^{(1)}$**:** Accepts non-empty words in which no two consecutive letters are the same.
- $\mathcal{A}^{(2)}$**:** Accepts words of odd length where every letter at an odd position of w is the letter a.

Hence,

$$L\big(A^{(2)}\big) = \big\{\, w \in \{a, b\}^* \;\mid\; |w| \text{ is odd and the odd-indexed letters are all } a \big\}.$$

b. Computation: $\widehat{\delta}^{(1)}\big(1, \texttt{abaa}\big)$

The automaton $A^{(1)}$ has states $\{1, 2, 3, 4\}$, where:

- State 1 is the initial state (non-final).
- State 2 indicates the last letter read was a (no violation).
- State 4 indicates the last letter read was b (no violation).
- State 3 is the "sink" or "trap" state (once a violation occurs, e.g. two consecutive letters the same).

We compute step-by-step for the input `abaa`:

$$\widehat{\delta}^{(1)}(1, \texttt{abaa}) :$$

1. Start in state 1, input = `abaa`.
2. Read 'a': $\delta^{(1)}(1, a) = 2$. Remaining input = `baa`.
3. Now in state 2, read 'b': $\delta^{(1)}(2, b) = 4$. Remaining input = `aa`.

4. Now in state 4, read ‘a’: $\delta^{(1)}(4, a) = 2$. Remaining input = `a`.
5. Now in state 2, read ‘a’: $\delta^{(1)}(2, a) = 3$. Remaining input is empty.

Therefore,

$$\widehat{\delta}^{(1)}(1, \texttt{abaa}) \;=\; 3.$$

Since state 3 is the non-accepting sink, the string `abaa` is not accepted by $A^{(1)}$.

b. Computation: $\widehat{\delta}^{(2)}\big(1, \texttt{abba}\big)$

Automaton $A^{(2)}$ can be thought of as having states:

- State 1: an even number of letters read so far (initial, non-final).
- State 2: an odd number of letters read so far, with “odd positions must be a” still satisfied (accepting).
- State 3: dead/trap state (if the condition on odd positions is violated).

We compute for `abba` (positions are 1,2,3,4):

$$\widehat{\delta}^{(2)}(1, \texttt{abba}) :$$

1. Start in state 1, input = `abba`.
2. Read ‘a’ (position 1): $\delta^{(2)}(1, a) = 2$. Remaining input = `bba`.
3. Now in state 2, read ‘b’ (position 2): $\delta^{(2)}(2, b) = 1$. Remaining input = `ba`.
4. Now in state 1, read ‘b’ (position 3): $\delta^{(2)}(1, b) = 3$. Remaining input = `a`.
5. Now in state 3, read ‘a’: $\delta^{(2)}(3, a) = 3$. Remaining input is empty.

Hence,

$$\widehat{\delta}^{(2)}(1, \texttt{abba}) = 3,$$

and state 3 is non-accepting. Therefore, `abba` is not accepted by $A^{(2)}$.

Problem 2: Product automata

We now define a *product automaton* A from $A^{(1)}$ and $A^{(2)}$ in order to recognize

$$L\big(A^{(1)}\big) \;\cap\; L\big(A^{(2)}\big).$$

a. Construct the intersection automaton A

States. The state set of A is the Cartesian product

$$Q \;=\; Q^{(1)} \;\times\; Q^{(2)}.$$

If $Q^{(1)} = \{1, 2, 3, 4\}$ and $Q^{(2)} = \{1, 2, 3\}$, then

$$Q = \{(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3), (4, 1), (4, 2), (4, 3)\}.$$

Initial state. $(1, 1)$, since 1 is the initial state of $A^{(1)}$ and 1 is the initial state of $A^{(2)}$.

Transition function. For $(p, q) \in Q$ and $x \in \{a, b\}$,

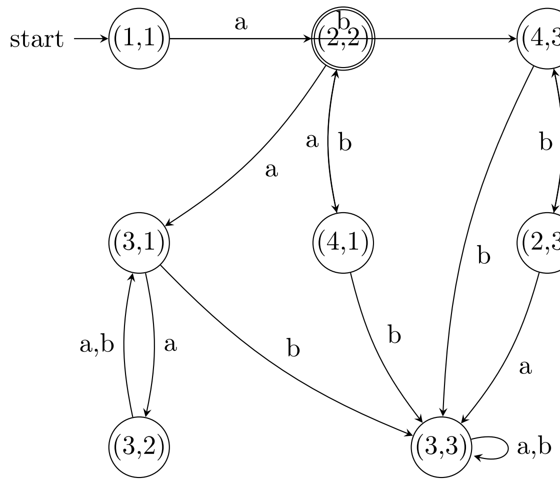
$$\delta\big((p, q), x\big) \;=\; \big(\delta^{(1)}(p, x),\; \delta^{(2)}(q, x)\big).$$

Final states (for intersection). A pair (p, q) is final if $p \in F^{(1)}$ and $q \in F^{(2)}$. - For $A^{(1)}$, we have $F^{(1)} = \{2, 4\}$ (all non-empty strings with no two consecutive letters the same). - For $A^{(2)}$, we have $F^{(2)} = \{2\}$ (odd-length strings with a in every odd position).

Thus

$$F \;=\; \{(p, q) \mid p \in \{2, 4\},\, q = 2\}.$$

Diagram of product automaton A showing all transitions. For brevity, not every state is drawn if some are unreachable.



b. Why $L(A) = L\big(A^{(1)}\big) \cap L\big(A^{(2)}\big)$

A string w is accepted by A precisely if:

- Following the transitions of A on w leads from $(1, 1)$ to a final state (p, q).
- This happens exactly when p is a final state of $A^{(1)}$ *and* q is a final state of $A^{(2)}$.

Thus w is accepted by both $A^{(1)}$ and $A^{(2)}$. Hence $L(A) = L(A^{(1)}) \cap L(A^{(2)})$.

c. Constructing A' for the Union

To get $L(A') = L\big(A^{(1)}\big) \cup L\big(A^{(2)}\big)$, we keep the same product structure but change the final states. A product state (p, q) is final if *either* $p \in F^{(1)}$ *or* $q \in F^{(2)}$. Formally:

$$F' \;=\; \big(F^{(1)} \times Q^{(2)}\big) \;\cup\; \big(Q^{(1)} \times F^{(2)}\big).$$

This ensures that a string is accepted by A' if it is accepted by $A^{(1)}$ *or* $A^{(2)}$.

Summary of Key Points:

- For $A^{(1)}$, the accepting states are $\{2, 4\}$; consecutive letters must differ and the string must be non-empty.
- For $A^{(2)}$, the accepting state is $\{2\}$; the string must have odd length and a in every odd position.
- Extended transitions showed that $\widehat{\delta}^{(1)}(1, \texttt{abaa}) = 3$ (not accepted) and $\widehat{\delta}^{(2)}(1, \texttt{abba}) = 3$ (not accepted).
- The product automaton for intersection has final states $F^{(1)} \times F^{(2)} = \{(2, 2), (4, 2)\}$, while for union we use $F' = (F^{(1)} \times Q^{(2)}) \,\cup\, (Q^{(1)} \times F^{(2)})$.

Exercise 2.2.7: Induction Proof

Statement: Let A be a DFA, and let q be a particular state of A such that

$$\delta(q, a) \;=\; q \quad \text{for all input symbols } a.$$

Show by induction on the length of the input that for all input strings w, we have

$$\widehat{\delta}(q, w) \;=\; q.$$

Proof (by induction on the length of w):

Base Case ($|w| = 0$): If w is the empty string ϵ, then by definition of the extended transition function,

$$\widehat{\delta}(q, \epsilon) \;=\; q.$$

Hence the statement holds for $|w| = 0$.

Inductive Step: Assume that for all strings x of length n, we have

$$\widehat{\delta}(q, x) \;=\; q.$$

We need to prove the statement for any string w of length $n+1$. Let w be such a string. We can write w as

$$w \;=\; x\,a,$$

where x is a string of length n, and a is a single input symbol. Then,

$$\widehat{\delta}(q, w) \;=\; \widehat{\delta}(q,\, x\,a) \;=\; \delta\big(\widehat{\delta}(q, x),\, a\big).$$

By the inductive hypothesis, $\widehat{\delta}(q, x) = q$. Therefore,

$$\widehat{\delta}(q, w) \;=\; \delta(q, a).$$

But we are given that $\delta(q, a) = q$ for all symbols a. Hence,

$$\widehat{\delta}(q, w) \;=\; q.$$

This completes the inductive step.

Conclusion: By the principle of mathematical induction, for all strings w, we have

$$\widehat{\delta}(q, w) = q.$$

Thus, if a state q transitions to itself on every symbol, it remains q under any input string.

ITALC 2.3 Question: In the subset construction for converting an NFA to a DFA, often there is an exponential blow-up in the number of states. Can you propose a scenario for which this blow-up actually happens, and whether there are any strategies or special cases that might mitigate this worst-case behavior?