Algorithm Analysis and Automata

UT Knoxville, Spring 2023, COSC 312

Dr. Michael W. Berry, Alex Zhang

January 31, 2023

Contents

1 Introduction

As computer scientists, we like to think of ourselves as problem solvers. We can abstract problems in three distinct components: unknowns, data, and conditions.

In understanding how we abstract problems, computer scientists have conceptualized the Theory of Computation (TOC) which covers three areas:

- **Automata** – problem solving devices
- **Computability** – framework that categorizes devices by computing power
- **Complexity** – space complexity of tools used to solve them

In understanding our approach to solving problems, we think of the data, conditions, and unknowns as such:

- **Data** exists as “words” in a given “alphabet”
- **Conditions** form a set of words called a **language**
- **Unknowns** are boolean values, true if a word is in the language, false if otherwise

We denote an “alphabet” using Σ and all possible words of finite length with Σ^*. Any subset $L \subseteq \Sigma^*$ is a **formal language**.

2 Finite Automata

Definition 2.0.1 ▶ Finite Automaton

A **finite automaton** (FA) is a simple, idealized machine used to recognize patterns within input taken from some character set.

Finite automata can be used to generalize many applications ranging from parsers for compilers, pattern recognition, speech processing, and market prediction. We can formally define any finite automaton as a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite set of all possible states
- Σ is a finite set of symbols (alphabet)
- δ is a function $\delta : Q \times \Sigma \to Q$ (describes how input symbols change the state)
- $q_0 \in Q$ is the initial state
- $F \subseteq Q$ is the set of acceptable final states

Let's consider a simple, arbitrary finite automaton M_1.

- M_1 receives input symbols one at a time
- M_1 transitions states based on input
- When M_1 reads the last symbol, it outputs whether or not it ends in an acceptable final state

Definition 2.0.2 ▶ Acceptance

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and $w = a_1 \dots a_n$ be a string over Σ. We say M **accepts** w if there exists a sequence of states $r_0 \dots r_n$ in Q such that:

- $r_0 = q_0$
- $\delta(r_i, a_{i+1}) = r_{i+1}$
- $r_n \in F$

Finite automaton design approach:

1. Identify the possible states of the finite automaton
2. Identify the condition to change from one state to another
3. Identify initial and final states
4. Add missing transitions

Given a finite automaton, we can deduce the language of possible inputs.

Example 2.0.1 ▶ Simple Finite Automaton

Let $M_1 \coloneqq (Q, \Sigma, \delta, q_1, F)$ where $Q = \{q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, and $F = \{q_2\}$. Define a possible transition function δ.

The transition function $\delta : Q \times \Sigma \to Q$ must map every ordered pair of a state and letter to another state.

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

Then, the language is:

$$L(M_1) = \{(w \in \Sigma^* : 1 \in w) \wedge (\text{has even number of 0s following the last 1})\}$$

Example 2.0.2 ▶ Simple Finite Automaton

Let $M_2 \coloneqq (Q, \Sigma, \delta, q_1, F)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{0, 1\}$, and $F = \{q_2\}$. Consider a possible transition function δ defined as such:

	0	1
q_1	q_1	q_2
q_2	q_1	q_2

This time, our language is much simpler:

$$L(M_2) = \{w \in \Sigma^* : w \text{ ends in a } 1\}$$

Now imagine if kept everything the same but made $F = \{q_1\}$. Because our finite automaton's initial state is q_1, we must now consider the possibility of the empty word. Then our language is:

$$L(M_2) = \{w \in \Sigma^* : w \text{ does not end in } 1\}$$

Example 2.0.3 ▶ Pattern Recognizing Finite Automaton

Let $M_3 \coloneqq (Q, \Sigma, \delta, q_1, F)$ where $Q = \{s, q_1, q_2, r_1, r_2\}$, $\Sigma = \{a, b\}$, and $F = \{q_1, r_1\}$. Consider a possible transition function δ defined as such:

	0	1
s	q_1	r_1
q_1	q_1	q_2
q_2	q_1	q_2
r_1	r_2	r_1
r_2	r_2	r_1

Now our machine encompasses two smaller machines. s acts as a branch point where we must lock ourselves into either q states or r states.

Then our language is:

$$L(M_3) = \{w \in \Sigma^* : (w \text{ starts and ends with } a) \vee (w \text{ starts and ends with } b)\}$$

Now, let's start with a given language, then find an acceptable transition function δ.

Example 2.0.4 ▶ Finding δ from Language

Suppose $\Sigma = \{a, b\}$. Let F_1 be a finite automaton that recognizes the language $A_1 \coloneqq \{w : w \text{ has at exactly two a's}\}$. Let F_2 be a finite automaton that recognizes the language $A_2 \coloneqq \{w : w \text{ has at least two b's}\}$.

3

Regular Languages

Definition 3.0.1 ▶ Regular Language

A language is **regular** if there exists a finite automaton that can accept every possible word from the language.

Definition 3.0.2 ▶ Concatenation

Let A and B be languages. A **concatenate** B is defined as:

$$A \circ B := \{xy | x \in A \wedge y \in B\}$$

Definition 3.0.3 ▶ Star

$$A^* := \{x_1 \dots x_k \,:\, k \geq 0, x_i \in A, 0 \leq i \leq k\}$$

Theorem 3.0.1 ▶ Closure of Regular Languages

Class of regular languages is closed under intersection and closed under complementation.

3.1 Nondeterminism

For complicated languages, it is difficult to create a completely deterministic finite automaton. If we forego determinism, we can have a more generalized finite automaton that allows for branching options.

Definition 3.1.1 ▶ Nondeterministic Computation

A machine that is **nondeterministic** is allowed to choose its next state.

To introduce nondeterminism, we need more choices for the next state and allow state change without any input.

Definition 3.1.2 ▶ Nondeterministic Finite Automaton (NFA)

A **nondeterministic finite automaton** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite set of states
- Σ is a finite set of symbols (alphabet)
- δ is a **relation** between $Q \times (\Sigma \cup \{\varepsilon\})$ and $\mathcal{P}(Q)$
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accept states

In this context, ϵ represents a nondeterministic choice by the machine.

Notice that in an NFA, we only specify that δ is a relation. NFA computation is usually more expensive than DFA computation. In theory, every NFA can be converted into an equivalent DFA.

Definition 3.1.3 ▶ Acceptance (NFA)

Let $N := (Q, \Sigma, \delta, q_0, F)$ be an NFA and $w := y_1 \dots y_n$ be a string over $\Sigma \cup \{\varepsilon\}$. We say N **accepts** w if there exists a sequence of states $r_0, \dots, r_m \in Q$ such that:

1. $r_0 = q_0$
2. $\delta(r_i, y_{i+1}) = r_{i+1}$ for $i = 0, \dots, m-1$
3. $r_m \in F$.

Theorem 3.1.1 ▶ Closure of Regular Languages

The class of regular languages is closed under the union operation.

Proof sketch. TODO: draw proof sketch here □

3.2 DFA/NFA Equivalence

When converting from a DFA to an NFA, we need to account for the machine's options.

Technique 3.2.1 ▶ Converting NFA to DFA

To convert an NFA to a DFA, we carry out the following steps:

1. Let $N := (Q, \Sigma, \delta, q_0, F)$ be an NFA that recognizes the language A.
2. Construct the DFA M that also recognizes A. For convenience, define $M := (Q', \Sigma, \delta', {q_0}', F')$.

3. For all $R \subseteq Q$, define $E(R)$ to be the collection of all states that can be reached from R by going along ϵ transitions, including members of R themselves.
4. Modify δ' to place additional edges on all states that can be reached by going along ϵ edges after every step.
5. Set ${q_0}' = E(\{q_0\})$ and $F' = \{R \in Q' : R \text{ contains an accept state of } N\}$.

Index

Definitions

Examples

Techniques

Theorems