Algorithm Analysis and Automata

UT Knoxville, Spring 2023, COSC 312

Dr. Michael W. Berry, Alex Zhang

February 9, 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.

Definition 2.0.2 ▸ Deterministic Finite Automaton (DFA)

A ***deterministic finite automaton*** is a finite automaton where each state transition leads to exactly one state.

We can formally define a DFA 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 called an ***alphabet***
- $\delta : Q \times \Sigma \to Q$ is a ***transition function***
- $q_0 \in Q$ is the ***start state***
- $F \subseteq Q$ is the set of ***accept 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 2.0.3 ▸ 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 $M_1 := (Q, \Sigma, \delta, q_1, F)$ 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 $M_2 := (Q, \Sigma, \delta, q_1, F)$ 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 $M_3 := (Q, \Sigma, \delta, q_1, F)$ 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 $A_1 := \{w : w \text{ has at exactly two a's}\}$. Let F_2 be a finite automaton that recognizes the language $A_2 := \{w : w \text{ has at least two b's}\}$.

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:

$$A \circ B \coloneqq \{xy | x \in A \land y \in B\}$$

Definition 3.0.3 ▶ Star

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

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 finite automaton where each state transition can lead to an arbitrary number of states, chosen by the machine.

We can formally define an NFA as a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite set of states
- Σ is a finite set of symbols called an ***alphabet***
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$ is a ***transition function***
- $q_0 \in Q$ is the ***start state***
- $F \subseteq Q$ is the set of ***accept states***

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

To account for the nondeterminism, δ now points to a subset of Q. 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 $N \coloneqq (Q, \Sigma, \delta, q_0, F)$ be an NFA and $w \coloneqq y_1 \dots y_n$ be a string over $\Sigma \cup \{\epsilon\}$. 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

The ambiguity in an NFA lies in the state transitions. We can convert an NFA into an equivalent DFA, shifting the ambiguity to the states.

Technique 3.2.1 ▸ Converting NFA to DFA

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

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

$$M \coloneqq (Q', \Sigma, \delta', q_0', F')$$

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 $F' = \{R \in Q' : R \text{ contains an accept state of } N\}$.

4 Irregular Languages

Definition 4.0.1 ▸ Regular Expression (RE)

We say that R is a ***Regular expression*** if R is one of the following:

- a for some $a \in \Sigma$ (a symbol of the alphabet)
- ϵ (language contains only the empty string)
- $\varnothing$ (language contains no strings)
- $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions
- $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions
- R_1^*, where R_1 is a regular expression

Theorem 4.0.1

A language is regular if and only if some regular expression describes it.

Proof. If a language R is described by a regular expression, then A is recognized by an NFA. Thus, A must be regular. If the language R is regular, then it is recognized by a DFA from which a regular expression can be deduced. ▢

Example 4.0.1 ▸ Irregular Languages

Consider the following languages:

- $B = \{0^n 1^n : n \geq 0\}$ is not regular.
- $C = \{w : w$ has an equal number of 0s and 1s$\}$ is not regular. We would need infinite states to account for all possible inputs.
- $D = \{w : w$ has an equal number of 01 and 10 substrings$\}$ is regular. We can think of this as recording transitions between 0s and 1s, and vice versa. In this sense, every 01 must be eventually followed by a 10, and every 10 must be eventually followed by a 01.

4.1 Pumping Lemma

How do we prove that a language is irregular? We would have to prove that there does not exist any machine that recognizes that language.

Definition 4.1.1 ▸ Pumping, Pumping Length

All strings in a language can be ***pumped*** if they are as long or longer than a specified ***pumping length***.

In other words, every string contains a section that can be repeated any number of times, and the resulting string is still in the language. For example, we can write sqrt(sqrt(sqrt(...(sqrt(x))...))) in C, and it's still valid.

Theorem 4.1.1 ▸ Pumping Lemma

Let A be a regular language. There exists a pumping length p such that, for any string s of length p or more, s may be divided into three pieces, $s = x \circ y \circ z$ where all the following hold:

1. $x \circ y^* \circ z \in A$
2. $|y| > 0$
3. $|x \circ y| \leq p$

Proof sketch. As a guide, the proof follows this general form:

- Let $M := (Q, \Sigma, \delta, q_1, F)$ be a DFA that recognizes A. Let p be the number of states.
- Show that any string $s \in A$ where $|s| \geq p$ can be broken into $x \circ y \circ z$ satisfying the three conditions.
- If there are no strings in A of length p or more, then the lemma is true because all three conditions hold for all strings of length p or more.

□

A formal proof is as follows:

Proof. Let $M := (Q, \Sigma, \delta, q_1, F)$ be a DFA that recognizes A. Let $p := |Q|$. Let $s := s_1 s_2 \ldots s_n$ be a string over Σ with $n \geq p$ and $r = r_1 r_2 \ldots r_{n+1}$ be the sequence of states encountered while processing s. TODO: finish proof □

Technique 4.1.1 ▸ Proving a Language is Irregular

To prove that a language A is not regular, we:

1. Suppose for contradiction A is regular
2. Find $s \in A$ where $|s| \geq p$ that cannot be pumped; consider all the ways of dividing

s into $x \circ y \circ z$, and show that for each division, at least one of the conditions fail.

Example 4.1.1 ▸ Simple Pumping Lemma Example

Prove that the language $B \coloneqq \{0^n 1^n : n \geq 0\}$ is not regular.

We can show that any substring of B cannot be pumped by contradicting the first condition of the Pumping Lemma.

Proof. Suppose for contradiction B is regular. Let p be the pumping length for B. Let $s \coloneqq 0^p 1^p \in B$. Then $|s| = 2p > p$. By the Pumping Lemma, we can partition s into three substrings, say x, y, z, such that $x \circ y^* \circ z \in B$. Let $s' \coloneqq x \circ y \circ y \circ z$. WE will show $s' \notin B$. Consider the three possible cases for the contents of y:

1. $y \in 0^+$. Then. s' has more 0s than 1s. Thus, $s' \notin B$, contradicting the first condition of the Pumping Lemma.
2. $y \in 1^+$. Then, following the logic from the first case, this is not possible.
3. y consists of 0s and 1s. Then $y \circ y \notin 0^n 1^n$, so $s' \notin B$.

□

We can simplify the above proof by targeting condition 3 instead:

Proof. By the third condition of the Pumping Lemma, we have $|x \circ y| \leq p$. Hence, y could only contain 0s. □

Example 4.1.2 ▸ Another Pumping Lemma Example

Prove that the language $E \coloneqq \{0^i 1^j : i > j\}$ is not regular.

Proof. Suppose for contradiction E is regular. Let p be the pumping length for E, and let $s \coloneqq 0^{p+1} 1^p \in E$. Then $|s| = (p+1) + p > p$. By the Pumping Lemma, we can partition s into three substrings, say x, y, z, such that $x \circ y^* \circ z \in E$. Then, by the third condition, we have $|x \circ y| \leq p$. Note that there are $p+1$ number of 0s at the beginning of s. Thus, $x \in 0^*$ and $y \in 0^+$. Consider the case when $i = 0$. Then $x \circ y^+ \circ z \notin E$, leaving only $x \circ z$. However, $|y| > 0$ by the second condition, so $x \circ z$ would lose at least one 0. Thus, $x \circ z \notin E$, so $x \circ y^* \circ z \notin E$. Therefore, E is not regular. □

Index

Definitions

Examples

Techniques

Theorems