Intro to Complexity Theory

Kunwar Shaanjeet Singh Grover

Contents

Chapter 1

Introduction

1.0.1 Automata, Computability, and Compelxity

What are the fundamental capabilities and limitations of computers?

- Complexity Theory: What makes some problems harder than others?
- Computability Theory
- Automata theory: Deals with the definations and properties of mathematical models of computation

1.0.2 Mathematical notions and terminology

Strings and Languages We define an **alphabet** to be any nonempty finite set. The members of the alphabet are the **symbols** of the alphabet. Generally $\sum$ and Γ are used to designate alphabets.

A **string over an alphabet** is a finite sequence of symbols from the alphabet written next to one another and not seperated by commas.

Chapter 2

Regular Languages

2.1 Finite Automata

Defination 1. *A **finite automation** is a 5-tuple $(Q, \sum, \delta, q_0, F)$ where*

1. *Q is a finite set called the **states***
2. *$\sum$ is a finite set called the **alphabet***
3. *$\delta : Q \times \sum \to Q$ is the **transition function***
4. *$q_0 \in Q$ is the **start state**, and*
5. *$F \subseteq Q$ is the **set of accept states***

If A is the set of all strings that machine M accepts, we say that A is the **language of machine** M and write $L(M) = A$. We say that M **recognizes** A or that M **accepts** A.

A machine may accept several strings, but it always recognizes only one lnaguage.

Formal defination of Computation

Let $M = (Q, \sum, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \dots w_n$ be a string where each w_i is a member of the alphabet $\sum$. Then M **accepts** w if a sequence of states $r_0, r_1, \dots, r_n$ in Q exists with three conditions:

1. $r_0 = q_0$,
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \dots, n-1$, and
3. $r_n \in F$.

Condition 1 says that the machine starts in the start state, Condition 2 says that the machine goes from the state to state according to the transition function. Condtion 3 sayas that machine accepts its input if it ends up in accept state. We say that M **recognizes language** A if $A = \{w \mid Macceptsw\}$

Defination 2. *A language is called a* ***regular language*** *if some finite automaton recognizes it.*

2.1.1 The Regular Operations

Defination 3. *Let A and B be languages. We define the regular operations* ***union, concatenation,*** *and* ***star*** *as follows:*

- ***Union*** $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$.
- ***Concatenation:*** $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$.
- $A^* = \{x_1 x_2 \dots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$.

2.2 NonDeterminism

When the machine was in a given state and read the next input symbol, we knew what the next state would be, it is determined. We call this **deterministic** computation. In a **nondeterministic** machine, several choices may exist for the next state at any point. Nondeterminism is a generalization of determinism, so every deterministic finite automaton is automatically a nondeterministic finite automaton.

2.2.1 Formal defination of a nondeterministic finite automaton

Defination 4. *A* ***nondeterministic finite automaton*** *is a 5-tuple $(Q, \sum, \delta, q_0, F)$, where*

1. *Q is a finite set of states,*
2. *$\sum$ is a finite alphabet,*
3. *$\delta : Q \times \sum_\epsilon \to P(Q)$ is the transition function,*
4. *$q_0 \in Q$ is the start state, and*
5. *$F \subseteq Q$ is the set of accept states.*

The formal defination of computation for an NFA is similar to that or a DFA. Let $N = \{Q, \sum, \delta, q_0, F\}$ be an NFA and w a string over the alphabet $\sum$.

Then we say that N **accepts** w if we can write w as $w = y_1 y_2 \dots y_m$, where each y_i is a member of $\sum_\epsilon$ and a sequence of states $r_0, r_1, \dots, r_m$ exists in Q with three conditions

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

Condition 1 says that the machine starts out in the start state. Condition 2 says that state r_{i+1} is one of the allowable next states when N is in the state r_i and reading y_{i+1}. Observe that $\delta(r_i, y_{i+1})$ is the *set* of allowable next states and so we say that r_{i+1} is a member of that set. Finally, condition 3 says that the machine accepts its input if the last state is an accept state.

2.2.2 Equivalence of NFAs and DFAs

Theorem 1. *Every nondeterministic finite automaton has an equivalent deterministic finite automaton*

Proof idea: We keep track of of the current states and create a transition function based on that. If there are k states o the NFA, then it has 2^k subsets of states. To keep track of these states, the DFA will have 2^k states.

Corollary 2. *A language is regular if and only if some nondeterministic finite automaton recognizes it.*

2.2.3 Closure under the regular operations

Theorem 3. *The class of regular languages is closed under the union operation*

Proof: Let:

$$N_1 = (Q_1, \sum, \delta_1, q_1, F_1) \text{recognize } A_1, \text{ and}$$
$$N_2 = (Q_2, \sum, \delta_2, q_2, F_2) \text{recognize } A_2.$$

Construct $N = (Q, \sum, \delta, q_0, F)$ to recognize $A_1 \cup A_2$.

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$
   The states of N are all the states of N_1 and N_2, with the addition of a new start state q_0.

2. The state q_0 is the start state of N.
3. The set of accept states $F = F_1 \cup F_2$.
4. Define δ so that for any $q \in Q$ and any $a \in \sum_\epsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \\ \delta_2(q,a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \text{and} a = \epsilon \\ \phi & q = q_0 \text{and} a \neq \epsilon \end{cases}$$

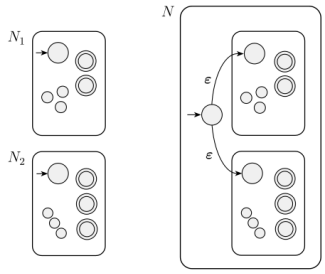

Figure 2.1: Construction of an NFA N to recognize $A_1 \cup A_2$

Theorem 4. *The class of regular languages is closed under the concatenation operation*

The formal proof is similar to the previous proof. The follwing image describes the contruction:

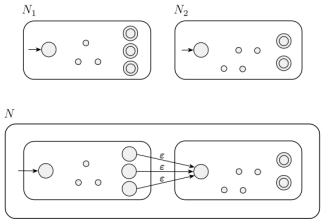

Figure 2.2: Construction of N to recognize $A_1 \circ A_2$

Theorem 5. *The class of regular languages is closed under the star operation.*

The formal proof is similar to union proof. The following image describes the construction:

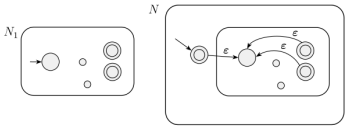

Figure 2.3: Construction of N to recognize A^*

2.3 Regular Expressions

In arthimetic, we can use operations $+$ and $\times$ to build up expressions such as

$$(5 + 3) \times 4$$

Similarly, we can use the regular operations to build up expressions describing languages, which are called **regular expressions**. Example:

$$(0 \cup 1)0^*$$

Defination 5. *Say that R is a **regular expression** if R is*

1. *a for some a in the alphabet $\sum$,*
2. *ϵ,*
3. *ϕ,*
4. *$(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,*
5. *$(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or*
6. *(R_1^*), were R_1 is a regular expression.*

2.3.1 Equivalence with finite automata

Theorem 6. *A language is regular if and only if some regular expression describes it.*

This theorem has two directions that we will prove.

Lemma 7. *If a language is described by a regular expression, then it is regular.*

Proof Idea: We can show how to convert R into an NFA recognizing A . By Corolary 2, if an NFA recofnizes A then A is regular. This can be done by proving that for each of the 6 cases of regular expressions, we can build a NFA for it.

Lemma 8. *If a language is regular, then it is described by a regular expression.*

Proof: We need to show that if a language A is regular, a regular expression describes it. Because A is regular, it is accepted by a DFA. We describe a procedure for converting DFAs into equivalent regular expressions.

We will first define a new type of finite automaton called a **generalized nondeterministic finite automaton**, GNFA. First we show how to convert DFAs into GNFAs, and then GNFAs into regular expressions.

GNFA are simply NDA wherein the transition arrows may have any regular expression as labels, instead of only members of the alphabet or ϵ.

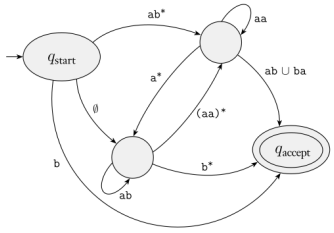

Figure 2.4: A generalized nondeterministic finite automaton

For convenience, we require that GNFAs always have a special form that meets the following conditions.

- The start state has transition arrows going to every other state but no arrows coming in from any other state.

- There is only a single accept state, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same at the start state.
- Except for the start and accept states, one arrow goes from the every state to every other state and also from each state to itself.

It is easy to convert a DFA into a GNFA in the special form. We simply add a new start state with an ϵ to the old start state and a new accept state with ϵ arrows from the old accept states. If any arrows have multiple labes, we take union of the previous labels. Finally, we add arrows labeled ϕ between states that had no arrows.

To convert GNFA into a regular expression, we take a k state GNFA, and form an equivalent $k-1$ state GNFA. If $k = 2$, The GNFA has a single arrow from accept state to the end state, the label of which is the required regular expression.

To construct a GNFA with one fewer state, we do this:

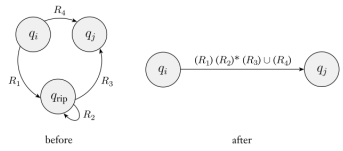

Figure 2.5: Constructing an equivalent GNFA with one fewer state

We do this for every pair $q_i.q_j$.
Formal Defination:

Defination 6. *A **generalized nondeterministic finite automaton** is a 5-tuple, $(Q, \sum, \delta, q_{start}, q_{accept})$, where*

1. *Q is the finite set of states,*
2. *$\sum$ is the input alphabet,*
3. *$\delta : (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \to R$ is the transition function,*
4. *q_{start} is the start state, and*
5. *q_{accept} is the accept state.*

2.4 Non-Regular Languages

Chapter 3

Context Free Languages

3.1 Context-Free Grammers

Example of a context-free grammer we will call G_1:

$$A \to 0A1$$
$$A \to B$$
$$B \to \#$$

Each grammer consists of a collection of **substitution rules**, also called **production**. Each rule appears as a line in the grammer, comprising a symbol and a string separeated by an arrow. The symbol is called a **variables**. The string consists of variables and other symbols called the **terminals**. Often the variable symbols are captial letters. terminals are similar to alphabet and are often represented by lowercase letters, numbers, or special symbols.

You use a grammer to describe a language by generating each string of that language as follows:

1. Write down the start variable. Usually the left-hand side top rule.
2. Find a variable written down and a rule starting witht hat variable. Replace the written down variable with the right-hand side of that rule.
3. Repeat step 2 until no varialbes remain.

For example,

$$A \to 0A1 \to 00A11 \to 000B111 \to 000\#111$$

All strings generated in this way constitute the **language of the grammer**.

3.1.1 Formal Defination Of A Context-Free Grammer

Defination 7. *A **context-free grammer** is a 4-tuple $(V, \sum, R, S)$, where*

1. *V is a finite set called **variables**.*
2. *$\sum$ is a finite set, disjoint from V, called the **terminals**,*
3. *R is a finite set of **rules**, with each rule beign a variable and a string of variables and terminals, and*
4. *$S \in V$ is the start variables*

If u, v and w are string of variables and terminals, and $A \to w$ is a rule of the grammer, we say that uAv **yields** uwv, written $uAv \implies uwv$. Say that u **derives** vm written $u \overset{*}{\implies}$, if $u = v$ or if a sequence $u_1, u_2, \ldots, u_k$ exists for $k \geq 0$ and

$$u \implies u_1 \implies u_2 \implies \cdots \implies u_k \implies v$$

The **language of the grammer** is $w \in \sum^* \mid S \overset{*}{\implies} w$