AN INTRODUCTION TO FINITE AUTOMATA

ARPIT MITTAL

Abstract. We discuss finite automata and their applications to the theory of generating functions. We also prove results related to finite automata such as the pumping lemma.

Contents

1. Machines

At its core, an automaton is a machine. The machine has states, and moves between them based on the input it receives. Suppose we have a light bulb. The light bulb can either be on or off. Thus, the light bulb has two states, *on* and *off*. The light bulb can turn on and off based on the direction of how the light switch is pulled. If the light bulb is on, and the switch is pulled up, the light bulb will remain to be on. On the contrary, if the light switch was pulled down, the bulb would be off. We can create a state diagram for the light bulb as follows.

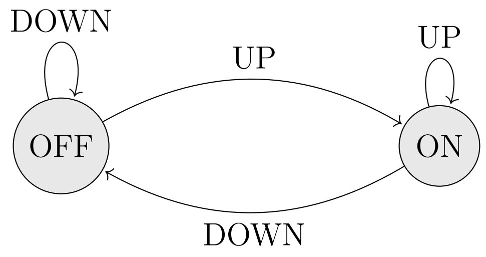

Figure 1

In the state diagram of an automata, the arrows between states are called transitions. In this state diagram, the directions "ON" and "OFF" refer to the direction of how the switch is pulled. For this to be a proper state diagram, it would need to have a start state. That is the state where the machine begins. Lets suppose that the bulb is off at the beginning, and only changes brightness if we tamper with the switch. An automata also needs accept states. The purpose of accept states is to determine whether a certain input is valid or not.

An input is only valid if the machine is at an accept state after the input is read. Say we want the light bulb to be on in the end. Then, we have the following state diagram.

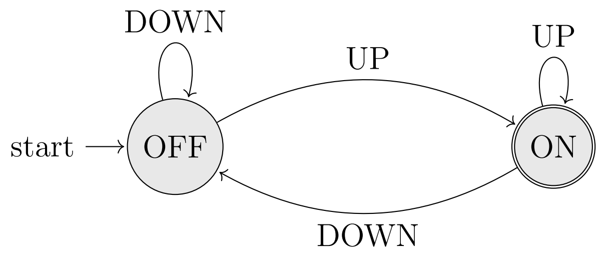

Figure 2

In the state diagram of an automata, the starting state has an arrow pointing towards it from the word "start" and the accepting states have a double circle wrapping them. Now that we have a proper state diagram for our machine, we can test it out with inputs. Suppose we get the input {UP, DOWN, DOWN, UP, UP, UP}. First, our machine is at OFF because it is the start state. Then, after receiving the input UP, the machine is at the state ON. After reading the next input, DOWN, the machine is back at the state OFF. We can keep reading inputs, and end up at the state ON. Since the state ON is an accept state, the machine accepts the input {UP, DOWN, DOWN, UP, UP, UP}.

2. Types of Finite Automata

The first type of a finite automaton is a Deterministic Finite Automaton (DFA).

Definition 2.1. A deterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

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

The transition function δ takes in a 2-tuple consisting of a state and an element in the alphabet, and returns a state.

Example 1

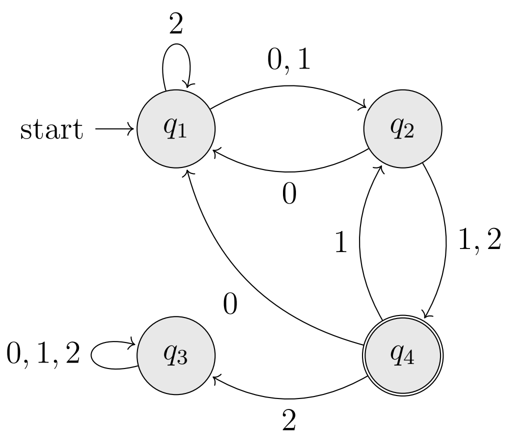

Figure 3

The formal description for this automaton is $(\{q_1, q_2, q_3, q_4\}, \{0, 1, 2\}, \delta, q_1, q_4)$. The transition function δ is

	0	1	2
q_1	q_2	q_2	q_1
q_2	q_1	q_4	q_4
q_3	q_3	q_3	q_3
q_4	q_1	q_2	q_3.

Suppose we get the input 0021212012. We start at q_1 and travel between the states in the following order: $q_1, q_2, q_1, q_1, q_2, q_4, q_2, q_4, q_1, q_2, q_4$. As q_4 is an accept state, the automaton does accept the input 0021212012.

Definition 2.2. If A is the set of all strings that the machine M accepts, then A is the language of machine M and $L(M) = A$. We say that M recognizes A.

Definition 2.3. A language is regular iff there exists a finite automaton that recognizes it.

Definition 2.4. A language is specification regular iff it has a specification in terms of the sum, product, sequence, set, multiset, and cycle constructions.

In all of the automata we have seen so far, there has been an outwards arrow from each state for each element in the alphabet. In other words, we always have a state to go to after reading the next character of an input, regardless of the current state. This characteristic of DFAs does not hold for a nondeterministic finite automaton (NFA). A NFA can have zero, one, or many outwards arrows from a state for a single element in the alphabet!

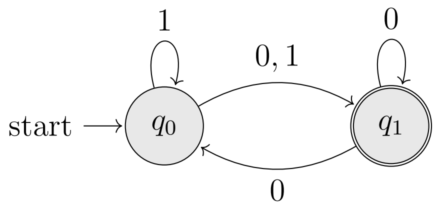

Figure 4

Figure 4 is an example of a NFA because it has multiple outwards arrows from a state containing the same element of the alphabet. Lets now see how an NFA computes. Suppose we get the input 101. We start at q_0 as it is the start state, and have 2 places to go after reading the first 1. We create a copy to go to state q_1 and remain at the state q_0 in the original machine. In our original machine, we go to q_1 after reading the 0 in the input string. In our copy, we create a copy of itself that goes to q_0 and go to q_1 in the original copy. In our original machine, we read the 1. There is no outwards arrow from q_1 with 1, so the original machine dies. In our first copy, it has nowhere to go at after reading the 1 so it dies. The second copy reads the 1 and creates a copy to go to q_1 while staying at q_0 itself. Since we have finished reading the input, and one of the copies is at an accept state, the NFA in Figure 4 accepts the input 101.

In general, when a NFA has multiple states to go to after reading an input, it creates copies of itself to go to each possible state. If there is nowhere to go after reading an input, that copy dies and is forgotten about. The NFA accepts the input if at least one of the copies ends in an accept state. NFAs have a special characteristic that makes them very powerful, the epsilon arrow. When a NFA is at a state that has an epsilon arrow going out of it, the NFA immediately creates a copy that follows the epsilon arrow. The NFA then carries out its functions normally. Let's look at an example.

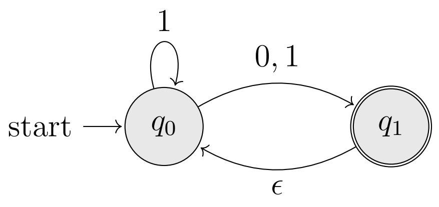

Figure 5

Suppose this NFA receives the input 10. The automaton first makes a copy that goes to q_1 and remains at q_0 in the original one. The original machine at q_0 reads the 0 and goes to q_1. The copy sees that there is an epsilon arrow and makes a copy of itself to go to q_0. The original copy then reads the 0 and dies. The copy at q_1 reads the 0 and goes to q_1. Since two of the copies ended at q_1 the NFA accepts the string 10.

If we let the power set of a set Q be $\mathcal{P}(Q)$ and Σ_ϵ be $\Sigma \cup \{\epsilon\}$ for any alphabet Σ, we can formally define NFAs as follows.

Definition 2.5. A nondeterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

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

Note that the codomain of δ is $\mathcal{P}(Q)$ rather than Q because a NFA can go to multiple states in Q rather than a single state after reading an input.

Definition 2.6. Two automatons are equivalent if they recognize the same language.

Proposition 2.7. *There is an equivalent deterministic finite automaton for every nondeterministic finite automaton.*

Proof. Let the NFA $N = (Q, \Sigma, \delta, q_0, F)$ recognize the language A. We want to construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ that also recognizes A. We will first ignore the epsilon arrows, and take them into account later. To create a DFA that is equivalent to a given NFA we need the DFA to keep track of all the possible states the NFA could be in at any time. Thus, we have $Q' = \mathcal{P}(Q)$. For $R \in Q'$ and $a \in \Sigma$, we can define

$$\delta(R, a) = \bigcup_{r \in R} \delta(r, a).$$

We thus must have $q_0' = \{q_0\}$ and $F' = \{R \in Q' | \text{R contains an accept state of N}\}$. Now, we can take the possible epsilon arrows in N into account by defining for $R \subseteq Q$,

$E(R) = \{q \in Q | \text{q can be reached by starting at R and traveling along one more epsilon arrows}\}.$

We can use this new function to modify δ' as follows:

$$\delta(R, a) = \bigcup_{r \in R} E(\delta(r, a)).$$

We can now change the start state of M to be $q_0' = E(q_0)$ which completes the proof. ■

Since we have shown that each NFA has an equivalent DFA, a very strong corollary can be made.

Corollary 2.8. *A language is regular iff it is recognized by a nondeterministic finite automaton.*

3. Regular Operations

Definition 3.1. We can define the following regular operations for languages A and B as follows:

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

Note that A^* is the set of any number of elements of A concatenated. Because you can concatenate 0 elements of A, it is always true that the empty string, $\epsilon \in A^*$.

Lemma 3.2. *If A and B are regular languages, $A \cup B$ is a regular language.*

Proof. Suppose the NFA N_1 accepts A and the NFA N_2 accepts B. We wish to construct a NFA N that recognizes $A \cup B$. The NFA N will accept an input if it is accepted by either N_1 or N_2. We construct N to have a new start state, and draw epsilon arrows from the new start state to the start states of N_1 and N_2. So when N is fed an input, it will go through both N_1 and N_2 and will be accepted if it ends at an accept state of either N_1 or N_2.

Let $N_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ and accept the language A. Let $N_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$ and accept the language B. We construct an automaton $N = (Q, \Sigma, \delta, q_0, F)$ that recognizes $A \cup B$.

(1) $Q = \{q_0\} \cup Q_1 \cup Q_2$.
(2) $\Sigma = \Sigma_1 \cup \Sigma_2$.

(3) For any $q \in Q$ and $a \in \Sigma_\epsilon$,

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

(4) $F = F_1 \cup F_2$.

■

Example. If the automaton in Figure 4 recognizes the language A, and the automaton in Figure 5 recognizes the language B, we would construct the following automaton to recognize $A \cup B$.

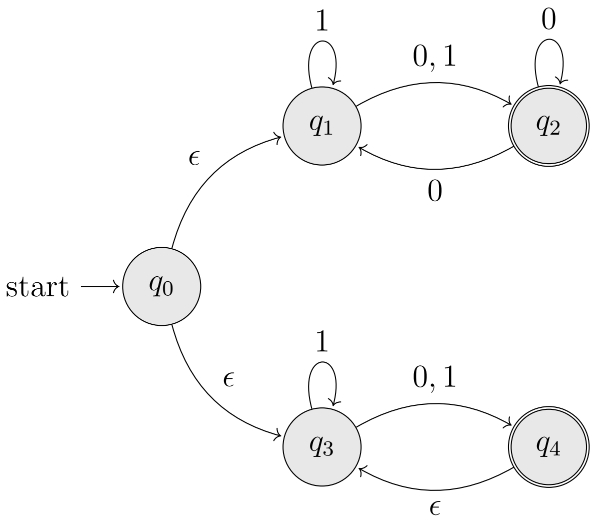

Figure 6

Lemma 3.3. *If A and B are regular languages, $A \circ B$ is a regular language.*

Proof. Suppose the NFAs N_1 and N_2 each recognize the languages A and B respectively. We wish to construct an automaton N which recognizes $A \circ B$. To do this we design N to first feed the input through N_1 and then into N_2. We can thus make the start state of N the start state of N_1 and the accepting states of N the accepting states of N_2. However, we must determine when to transfer the input from N_1 to N_2. For example, if the accepting states of N_1 each had transition arrows to some other states, and we drew a transition arrow from each accept state of N_1 to the start state of N_2 labeled by each element in the alphabet, it is possible that the input will leave N_1 too early and will be rejected by N_2. To solve this problem, we draw an epsilon arrows from the accept states of N_1 to the start state of N_2 so N will test all possible points of dividing the input into a piece for N_1 and a piece for N_2.

Let the NFAs $N_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ and $N_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$ recognize the languages A and B respectively. We construct an automaton $N = (Q, \Sigma, \delta, q_1, F_2)$ that accepts $A \circ B$.

(1) $Q = Q_1 \cup Q_2$.
(2) $\Sigma = \Sigma_1 \cup \Sigma_2$.

(3) For any $q \in Q$ and $a \in \Sigma_\epsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{if } q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q,a) & \text{if } q \in F_1 \text{ and } a \neq \epsilon \\ \{q_2\} \cup \delta_1(q,a) & \text{if } q \in F_1 \text{ and } a = \epsilon \\ \delta_2(q,a) & \text{if } q \in Q_2. \end{cases}$$

■

Example. If the automaton in Figure 4 recognizes the language A, and the automaton in Figure 5 recognizes the language B, we would construct the following automaton to recognize $A \circ B$.

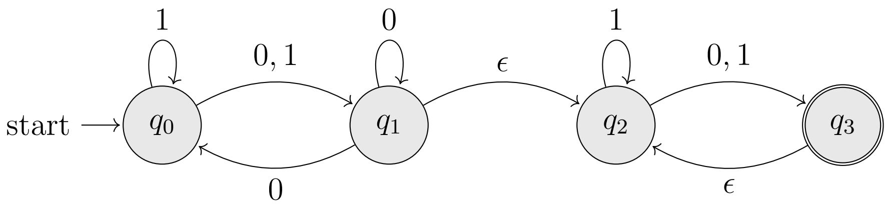

Figure 7

Lemma 3.4. *If A is a regular language, A^* is a regular language.*

Proof Idea. Suppose the NFA N_1 recognizes the language A. We wish to construct the NFA N that recognizes A^*. We construct N in a very similar way to how we constructed N in lemma 3.3. We must draw an epsilon arrow from each accept state of A to the start state of N_1 when we construct N. Since $\epsilon \in A^*$, we must create a new start state that is also an accept state and draw an epsilon arrow from that state to the original start state. We do this so ϵ will be accepted by N.

Let the NFA $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize the language A. We construct an automaton $N = (Q, \Sigma, \delta, q_0, F)$ that recognizes A^*.

(1) $Q = \{q_0\} \cup Q$.
(2) For any $q \in Q$ and $a \in \Sigma_\epsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{if } q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q,a) & \text{if } q \in F_1 \text{ and } a \neq \epsilon \\ \delta_1(q,a) \cup \{q_1\} & \text{if } q \in F_1 \text{ and } a = \epsilon \\ \{q_1\} & \text{if } q = q_0 \text{ and } a = \epsilon \\ \emptyset & q = q_0 \text{ and } a \neq \epsilon. \end{cases}$$

(3) $F = \{q_1\} \cup F_1$.

■

Example. If the automaton in Figure 4 recognizes the language A, we would construct the following automaton to recognize A^*.

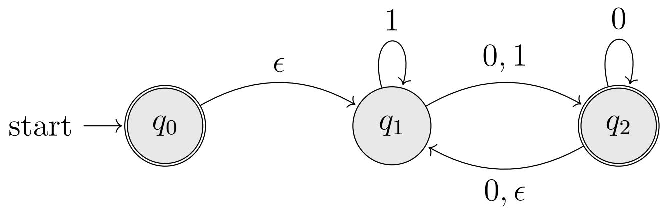

Figure 8

Theorem 3.5. *All three regular operations are closed on regular languages.*

Proof. This result follows from the previous lemmas. ■

4. Generating Functions

Theorem 4.1 (Kleene-Rabin-Scott)**.** *A language is specification regular iff there is some automaton that recognizes the language.*

We can also extract a generating function from a DFA.

Theorem 4.2. *Suppose that M is a deterministic finite automaton with a set of states Q, start state q_0, and accepting states $F \subseteq Q$. The generating function of $A = L(M)$ is determined under matrix form as*

$$A(z) = \mathbf{u}(I - zT)^{-1}\mathbf{v}.$$

In this theorem, I is the identity matrix and the transition function T is defined as

$$T_{j,k} = \#\{x \in \Sigma \textit{ such that an edge } (q_j, q_k) \textit{ is labeled by } x\}.$$

$\mathbf{u}$ *is the row vector $(1, 0, \dots, 0)$ and $\mathbf{v}$ is the column vector $(v_0, \dots, v_s)^t$ such that $v_j = [\![v_j \in F]\!]$. For predicate P, $[\![P]\!]$ is defined to be 1 if P is true, and 0 otherwise.*

We can go through an example from [FS09] of extracting an ordinary generating function from a deterministic finite automaton.

Example. Suppose we have a deterministic finite automaton with $\Sigma = \{a, b\}$ that recognizes words with the pattern *abb*. The state diagram for that automaton is as follows.

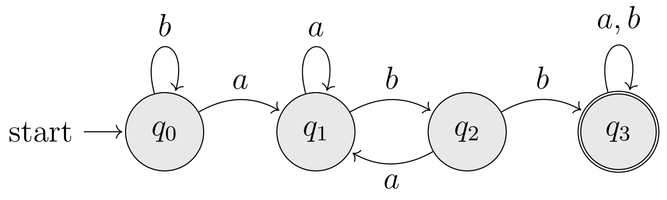

Figure 9

If we define $\mathcal{L}_j$ be the set of accepted words when starting at the state q_j, we have

$$\begin{aligned}
\mathcal{L}_0 &= a\mathcal{L}_1 + b\mathcal{L}_0\\
\mathcal{L}_1 &= a\mathcal{L}_1 + b\mathcal{L}_2\\
\mathcal{L}_2 &= a\mathcal{L}_1 + b\mathcal{L}_3\\
\mathcal{L}_3 &= a\mathcal{L}_3 + b\mathcal{L}_3 + \epsilon.
\end{aligned}$$

We can use this system of equations to obtain a system of equations for the ordinary generating functions.

$$\begin{aligned}
L_0 &= zL_1 + zL_0\\
L_1 &= zL_1 + zL_2\\
L_2 &= zL_1 + zL_3\\
L_3 &= zL_3 + zL_3 + 1.
\end{aligned}$$

We can solve this equation for

$$L_0(z) = \frac{z^3}{(1-z)(1-2z)(1-z-z^2)}.$$

We take the partial fraction decomposition from which we obtain

$$L_{0,n} = 2^n - f_{n+3} + 1,$$

where f_n is the nth Fibonacci number. It is also possible to avoid doing all of this work and just directly find the specification for the language. We can get the specification

$$\mathcal{L}_0 \cong \textsc{Seq}(b) \times a \times \textsc{Seq}(a) \times b \times \textsc{Seq}(a \times \textsc{Seq}(a) \times b) \times b \times \textsc{Seq}(a+b)$$

from which we can obtain the same generating function. We get this specification by observing how to get to state q_i from state q_{i-1}. Going back to our expression for $L_0(z)$, we can see how it is directly reflected in the visual state diagram. The state q_0 has a single self loop which is where the $(1-z)$ in the denominator of $L_0(z)$ comes from. The state q_3 has two single loops from which the $(1-z-z)$ comes from. The state q_1 has a single loop of a, and a double loop of going to q_2 and back to q_1 which gives the $(1-z-z^2)$.

5. The Pumping Lemma

The pumping lemma gives a quality of all regular languages.

Lemma 5.1 (Pumping Lemma). *If A is a regular language, there exists a number p (the pumping length) where if s is a string in A with length at least p, it can be divided into 3 pieces $s = xyz$, where*

(1) *for each $i \geq 0$, $xy^iz \in A$,*
(2) *$|y| > 0$, and*
(3) *$|xy| \leq p$.*

To prove this lemma, we will need to use the pigeonhole principle. The pigeonhole principle states that if there are n pigeons and $m < n$ holes, there will be at least one hole with more than one pigeon.

Proof. Let M be the deterministic finite automaton $(Q, \Sigma, \delta, q_1, F)$ recognizing the language A. Let $p = |Q|$. Now, we can let $s = s_1s_2s_3\dots s_n$ be a string in A and $r_1, r_2, \dots, r_{n+1}$ be the sequence of states M goes through while reading s where $n \geq p$. Note that the sequence of states has length $n+1$ rather than n because there is a start state, and then a state after each s_i. Since $n+1 \geq p+1 > p$, two states among the first $p+1$ states in the sequence must be the same. We label the first equivalent state as r_j and the second as r_k. We then break up s into xyz where

(1) $x = s_1, \dots, s_{j-1}$,
(2) $y = s_j, \dots, s_{k-1}$, and

(3) $z = s_k, \dots, s_n$.

Because x takes the automaton from the state r_1 to r_j, y takes the automaton from r_j to $r_k = r_j$, and z takes the automaton from r_k to r_{n+1}, condition 1 of the Pumping Lemma is satisfied. Condition 2 is trivial as $k \neq j$ which means that the smallest $|y|$ can be is 1 in the case of a self loop ($k = j + 1$). Condition 3 follows from the fact that $k \leq p + 1$. Thus, the proof is complete. ■

The pumping lemma can be used to prove that a language is non-regular by using a proof by contradiction. We can assume that the language is regular, and then show that there is no p for the conditions of the pumping lemma to be met. We can go through an example from [Sip12] to illustrate this method.

Example. Let A be the language $A = \{0^n 1^n | n \geq 0\}$. Assume for the sake of contradiction that A is regular. Let p be the pumping length of A. Since $s = 0^p 1^p$ is a string in A that has a longer length than p, it must be able to break down into xyz where $xy^iz \in A$ for all $i \geq 0$. We have three possibilities for y, so we must show that each possibility results in a violation of the condition.

(1) **Case 1:** The first possibility for y is that it consists of only zeroes. Thus, xy^iz for some $i \geq 1$ would have more zeroes than ones, meaning that it is not in A, violating the first clause of the pumping lemma.
(2) **Case 2:** The second possibility for y is that it consists of only ones. Similar to the first case, this would contradict clause 1 of the pumping lemma as xy^iz would have more ones than zeroes.
(3) **Case 3:** The third possibility for y is that it consists of both zeroes and ones. In this case, xy^iz would have the same number of zeroes as ones, but the zeroes and ones will be out of order and not all zeroes would be before the ones. Thus, xy^iz is not a member of A which contradicts the first clause of the pumping lemma.

Since the pumping lemma does not hold on this language, this language is not regular.

References

[FS09] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics.* Cambridge University Press, 2009.
[Sip12] Michael Sipser. *Introduction to the Theory of Computation, 3rd edition.* Cengage Learning, 2012.

Email address: `arpit.mittal.2.71@gmail.com`