CS 4510 Automata and Complexity January 8th 2024

Lecture 1: Introduction

Lecturer: Abrahim Ladha *Scribe(s): Akshay Kulkarni*

1 Introduction

Welcome to CS 4510: Automata & Complexity Theory. This course is primarily the study of two questions:

1. What are the limits of computation? This question is the study of computability theory. We are not concerned with efficiency, but simply possibility. The questions we can ask are so abstract, we can only really discuss if there are solutions at all. Most of the questions in this field have been solved in a dramatic fashion.
2. What makes some problems easy and others hard? This question is the study of complexity theory. Why do certain problems appear to require a certain amount of resource? Most of the questions in this field are unsolved. This contrast is what makes the course interesting.

1.1 Formal Language Theory

Before we can discuss these challenges, we need a syntax and system set us which allows us to discuss what a problem and solution are. To that end, we borrow elementary tools from linguistics and set theory.

Definition 1.1. An alphabet, denoted as Σ is a finite set of symbols.

Definition 1.2. A word, or string, is a finite sequence of symbols from some alphabet.

Examples:

1. $\Sigma = \{a, b\}, \{1\}, \{0, 1\}, \{a, \ldots, z, A \ldots, Z\}$
2. $\Sigma^2 = \Sigma \times \Sigma = \{aa, ab, ba, bb\}$ (strings of length 2) A string or word, is simply an ordered finite length sequence of symbols.
3. $\Sigma^n =$ strings of length n.
4. $\Sigma^0 = \{\varepsilon\}$ where ε is a special symbol we use to denote the only string of length zero. $\varepsilon =$"". Note that it is different than $\emptyset$. While ε is a word without length, $\emptyset$ is a set without elements, a collection without objects. They have a fundamentally different type. Σ^0 is then a set of one element, which is the word of no length.
5. We define Σ^* as a union like

$$\Sigma^* = \bigcup_{i=1}^{\infty} \Sigma^i = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots \tag{1}$$

Note that Σ^* contains every string. Every string has a length, and is then in some Σ^i, so is then in this union. Do not be seduced into thinking about words with infinite length. We will discuss why later, but for now, know that every word has finite length.

6. We may use the notation $a^5 = aaaaa$ for repetition of symbols. For example, $a(bc)^3d = abcbcbcd$ while $ab^3c^3d = abbbcccd$.

A *language* is any set of strings $L \subseteq \Sigma^*$. For example:

- $L = \{aa, bb, abab, aaa, b\}$
- $L_1 = \{w \in \Sigma^* \mid w \text{ begins with } a\}$
- $L_2 = \{w \in \Sigma^* \mid \#a(w) \text{ is even}\}$
- $L = \{a^n \mid n \text{ is even}\}$
- $L = \{w \in \Sigma^* \mid \#a(w) \equiv 3, 4 \pmod 7\}$
- $L = \{w \in \Sigma^* \mid \text{int}(w) \text{ is prime}\}$

1.2 Automata

An automata is a hypothetical model of a computer. We may study the limitations of certain automata, or compare them to one another. We do not really care about the automata themselves, but what they can tell us about the kinds of problems they can solve.

We need the ability to first discuss what it means to solve a problem, and here we borrow tools from formal language theory. A decision problem is a partition of Σ^* into the "good" and the "bad". We give an automata a word, and it will either accept the string or reject it. We say that an automata M decides a language L if:

$$M(w) \textit{ accepts} \iff w \in L \tag{2}$$
$$M(w) \textit{ rejects} \iff w \notin L \tag{3}$$

We are concerned with what kinds of automata can decide what kinds of languages.

There are two perspectives. First fix the machine, and note that each machine must define some language. Every machine has some behavior. Next is to fix the language and consider all possible machines which may decide it correctly. We are more concerned with the second perspective than the first.

2 Deterministic Finite Automata

Definition 2.1. A DFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$:

1. $Q = \{q_0, \ldots, q_k\}$ is a finite set called the *states*,
2. Σ is the finite *alphabet*, usually $\Sigma = \{a, b\}$ or $\{0, 1\}$
3. $\delta : Q \times \Sigma \to Q$ is the *transition function*. It is a well-defined finite function. Every state symbol pair in the input has a single output.

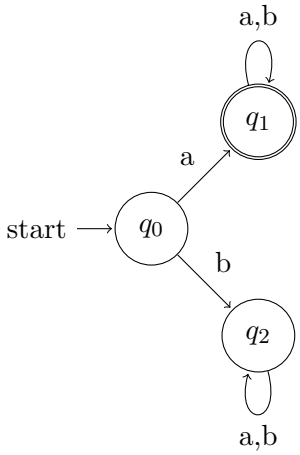
4. $q_0 \in Q$ is the designated *start state*. We have to start somewhere.

5. $F \subseteq Q$ is the set of *acceptance*, or final states. If a state is not final, we may say it is rejecting.

We define a computation of a DFA on a word to be a repeated sequence of applications of the transition function, one per letter in the word sequentially. We say the DFA accepts the word if the computation terminates on a final state, and the DFA rejects if the computation terminates on a non-final state.

2.1 Examples

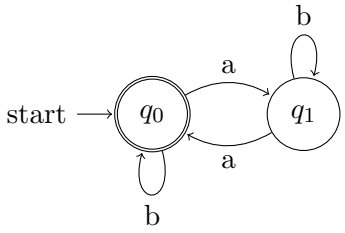
1. $L_1 = \{w \in \Sigma^* \mid w \text{ begins with } a\}$



Before discussion of this specific DFA, we note the notation of DFAs in general. The previous formal definition can be cumbersome, so it is better to give a *state diagram*. A state diagram is what you see above, sort of like a graphical programming language. We denote the start state as q_0 and with a tiny arrow from nothing. We denote an accepting state as one with a double circle, and a rejecting state as one without. Note that our transition function is the edges, and the function is well-defined when each state has $|\Sigma|$ outgoing transitions, one per symbol.

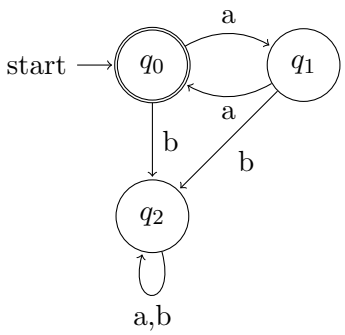
Consider a computation of this DFA on any word. It branches to two different states on the first letter. Once you enter states q_1 or q_2, you may not leave. Once you enter either of these two purgatories, the rest of the letters of the word are ignored. We denote q_1 as the good purgatory by making it a final state, and q_2 as the bad purgatory by making it a rejecting state.

2. $L_2 = \{w \in \Sigma^* \mid \#a(w) \text{ is even}\}$



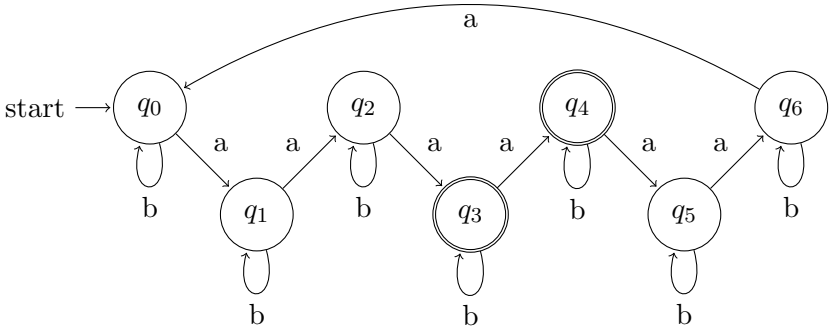
Here we have transitions to keep track modulo two the number of a's we have seen. We have self loops for the b's because we want to ignore them. What if we wanted to reject whenever we saw any b's, and not ignore them?

3. $L = \{a^n \mid n \text{ is even}\}$



Now as soon as we see a b, we immediately enter purgatory and can never leave.

4. $L = \{w \in \Sigma^* \mid \#a(w) \equiv 3, 4 \pmod 7\}$



There is nothing too special about the number two. We may generalize the previous example to keep track of residues modulo any other number. Create one state per equivalence class, and simply transition between them. Going from one state to the next means you have seen an additional a. Going around the clock means you have seen a seven times.

5. $L = \{w \in \Sigma^* \mid \text{int}(w) \text{ is prime}\}$
   We are not concerned with the alphabet really, not for the problems we want to study. Analogously, the primality of an integer has nothing to do with the base it is represented in. Numbers themselves are not real. Seventeen is an idea, a useful abstraction. If I have 17 bananas and I eat 15 of them, I have two bananas left. But this is a property of the quantity and not the bananas. The statement is true when discussing anything else. A number is just an idea, and not something tangible, but we still need a way to discuss these ideas, so we have to represent them somehow. But importantly, note that the way we represent them has little to do with the idea itself. 17 is prime, but so is "seventeen" and 10001_2. We use languages and strings just as a tool to talk about what we care about: decision problems. We have turned computational questions into ones about set membership. If an automata can determine correctly if some $w \in L$, then it certainly has the power of primality testing. Maybe it can then solve other problems related to the prime numbers. If we can show some other kind of machine cannot correctly decide this language, then maybe those related problems are also beyond its computational power.

3 Double Simulation

We may use one DFA to simulate two other DFAs simultaneously. Consider how DFAs are analogous to a very limited kind of program. Among its other limitations, it only uses constant memory. We may certainly combine two constant memory programs into one (bigger) constant memory program. This is the intuition. Each state of our new DFA will correspond to a pair of possible states in two different DFAs. Computation on our DFA will correspond to computation on two other DFAs in parallel.

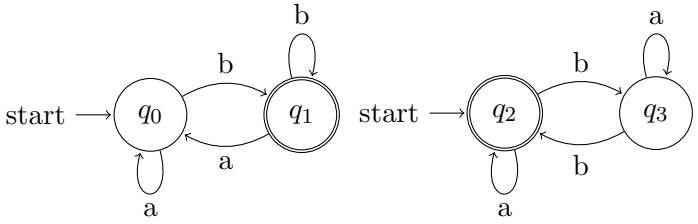
Let L_1 be decided by DFA $(Q_1, \Sigma, q_0^1, \delta_1, F_1)$ and L_2 be decided by DFA $(Q_2, \Sigma, q_0^2, \delta_2, F_2)$. We program a DFA $(Q, \Sigma, q_0, \delta, F)$ to decide $L_1 \cap L_2$ as follows:

1. $Q = Q_1 \times Q_2$
2. Σ is the same
3. $$\delta((q_i, q_j), a) = (\delta_1(q_i, a), \delta_2(q_j, a))$$
   for $q_i \in Q_1$ and $q_j \in Q_2$
4. $q_0 = (q_0^1, q_0^2)$
5. $F = F_1 \times F_2$

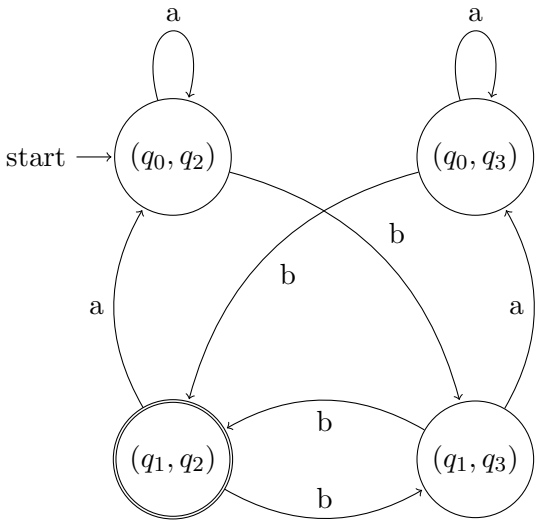
3.1 example

$$L_1 = \{w \in \Sigma^* \mid w \text{ ends with a } b\} \tag{4}$$
$$L_2 = \{w \in \Sigma^* \mid \#b(w) \text{ is even}\} \tag{5}$$

Lets make two DFAs for these languages.



Out cartesian product DFA then looks like the following:



We may assign meaning to the states. State (1,2) means being in state 1 in DFA1 and state 2 in DFA2 simultaneously. You may only end on state 1 if your string ends with a b, and you may only end on state 2 if you have seen an even number of b's at that point. So strings which end on state $(1, 2)$ are those which both end with b and have seen an even number of b's. Note that if we made $F_3 = (Q_1 \times F_2) \cup (F_1 \times Q_2) = \{(0, 2), (1, 2), (1, 3)\}$ we would have accepting states for the union of our two languages. The additional states correspond to accepting in either simulated DFAs, but not both.

3.2 Regularity

What kinds of languages can DFAs decide? We don't yet know the problems they are capable of solving or not solving. We say a language is regular if and only if it is decided by a DFA.

Definition 3.1. We write the set of languages decidable by a DFA as $\mathscr{L}(DFA)$. These are the regular languages.

Note that a word is a finite sequence of symbols, a language is a (possibily infinite) set of words, and a class is a (possibily infinite) set of languages. A class is a set of sets of strings.