Notes on the Theory of Computation

Part 1: Regular Languages

Coding Club

April 14, 2024

Mohammed Alshamsi
2021004826
mo.alshamsi@aurak.ac.ae

Department of Computer Science and Engineering
American University of Ras Al Khaimah
2023–24

Contents

List of Figures

1 Introduction

Welcome to the second set of "lecture notes" for the AURAK Coding Club. The idea is to introduce a useful topic that lends itself easily to coding applications. These topics are mathematical in nature, but most (if not all) of it will be intuitive material that doesn't require much background knowledge.

1.1 How to Read this Document

This is essentially just a short monologue about regular languages and related concepts. We'll go over the most basic definitions and results, and give some examples. This is by no means meant to give you a comprehensive introduction; it's just a quick tour to get you started. Check the bibliography if you want to learn more about the topic.

I'll be sure to bring up coding applications whenever it makes sense. Any code I write at those points will be in the C language. If you've taken (or are taking) CSCI 112, you'll be able to follow along without much trouble. Furthermore, there will be coding exercises. You're free to solve them in any language.

If you don't know any coding, check this link for a quick intro to C.

1.2 Basics

It takes a bit of time to explain what a *regular* language is, but the definition of a *language* is fairly simple.

Definition 1.1 (Alphabets).

An alphabet Σ is any nonempty finite set. The members of the alphabet are said to be the *symbols* of the alphabet.

Definition 1.2 (Strings).

A string w over Σ is a finite sequence of symbols from Σ.

$|w|$ denotes the *length* of w, which is the number of symbols it contains. w^R is the *reverse* of w. ϵ is the empty string, containing zero symbols, and is a string over all alphabets.

For two strings w and v, wv is the *concatenation* of the strings, and is the string obtained by "appending" v to the end of w.

w^n for some natural number n denotes the concatenation of n copies of w. Further, $w^0 = \epsilon$.

Definition 1.3 (Languages).

A language over Σ is *a* set of strings over Σ. The empty language, containing no strings, is denoted $\emptyset$. The *Kleene closure* of Σ, denoted Σ^*, is the set of *all* strings over Σ.

Example (Languages).

Suppose $\Sigma = \{a, b\}$. Examples of strings over this are ab, $baab$, $ababa$. The lengths of these strings are respectively 2, 4, and 5; and their reverses are respectively ba, $baab$, and $ababa$. A language is any set of strings composed of symbols from Σ:

1. The empty set $\{\}$ is a language over Σ. (And, in fact, over all alphabets.).
2. $\{aab, b\}$ is a language over Σ.
3. So is $\{a, aa, aaa, \ldots\}$.

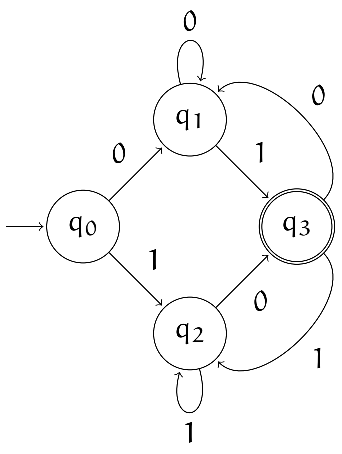

Figure 1: A rudimentary DFA.

2 Finite Automata

Moving forward, we'll discuss basic models of computation which *recognize* languages. Then, we'll discuss a few applications.

2.1 Deterministic Finite Automata

Take a look at Figure 1. A DFA is composed of several states, here q_0 through q_3. Start at q_0; this is the state you are currently on. (You can place your finger on it to signify this). You will have some input string, say 1110. Now notice that each state has two arrows from it: one labeled 0, and one labeled 1. For each character in the input string, move your finger to the state indicated by the arrow corresponding to that character. For 1110, you would first move to q_2 through the first 1. Then, you would remain in q_2 for the next two 1 inputs, because the arrow corresponding to input 1 is a loop back onto q_2. Finally, the 0 input will take you to q_3.

Notice that q_3 is circled twice — this indicates that it is a final state. If the input ends while your finger is here, then the string is said to be *accepted*. Otherwise, it is rejected. The set of all strings accepted by an automaton is said to be the language recognized by that automaton.

2.1.1 Formal Description

The formal definition will be necessary to speak precisely about automata.

Definition 2.1 (Deterministic Finite Automaton).

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

1. Q is a *set of states*.
2. q_0 is an element of Q known as the *starting state* or the *initial state*.
3. F is a subset of Q and is the *set of final states*.
4. Σ is the input alphabet.
5. δ is the *transition* or the *next-state* function, mapping $Q \times \Sigma$ to Q.[1]

[1]This maps a state-and-symbol pair to a state. It tells you the next state based on the current state and the input character.

Example.

For the DFA in Figure 1:

1. The set of states is $Q = \{q_0, q_1, q_2, q_3\}$
2. The initial state is q_0.
3. The set of final states is $\{q_3\}$.
4. The input alphabet Σ is $\{0, 1\}$.
5. δ can be represented by this table:

δ	0	1
q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_3	q_2
q_3	q_1	q_2

DFAs are usually the easiest to implement. Here's a C program for the DFA of Figure 1:

```c
#include<stdio.h>

int main() {
    int delta[4][2] = {
        { 1, 2 },
        { 1, 3 },
        { 3, 2 },
        { 1, 2 }};
    int currentState = 0;
    char input[512];

    printf("Input: ");
    scanf("%511[^\n]",input); // Read a full line, spaces included

    int index = 0;

    while (input[index] != '\0') { // while not end of string
        printf("Reading %c\n",input[index]);
        // set next state according to table
        currentState = delta[currentState][input[index] - '0'];
        index += 1;
    }

    if (currentState == 3) { // 3 is final state
        printf("String %s accepted\n",input);
    else {
        printf("String %s rejected\n",input);

    return 0;
}
```

This design can be generalized to any DFA; mainly, the changes would be to the δ table and to the method of detecting whether the state is a final state. For the latter, a more general approach would be to use an array for F, and to check whether the current state is in that array.

Exercise 1.

What language is recognized by the DFA in Figure 1? Remember that this is the set of all strings that are accepted by the automaton.

Exercise 2.

This one's a little tough, but not impossible. Construct a DFA which recognizes the language

$$L = \{\epsilon, 0^2, 0^3, 0^4, 0^6, 0^8, \ldots\},$$

which contains all strings of n zeroes, where n is divisible by 2 or by 3 (or both).

Give the 5-tuple $(Q, q_0, F, \Sigma, \delta)$ for this automaton, and implement it.

2.2 Nondeterministic Finite Automata

Now for the NFAs. There are two types: NFA, and NFA$_\epsilon$. For a standard NFA, its δ function is from $Q \times \Sigma$ to $\mathcal{P}(Q)$, which is the power set of Q.[2] This means that the same state-symbol combination can take you to multiple states, or even no states at all.

When tracing the behavior of an NFA with your hand, you won't be restricted to one finger like with DFAs. Whenever your finger is on a state q and you read an input character σ, you'll need to place a finger on each of the states in the set $\delta(q, \sigma)$; that is, every state that follows from q on input σ. When there are multiple states in $\delta(q, \sigma)$, this results in a form of "branching". When there is no transition from q on input σ — meaning $\delta(q, \sigma) = \emptyset$ — you'll simply remove the finger from q, since there is nowhere to move it to.

An NFA recognizes a string if at least one branch is on a final state when the input ends.

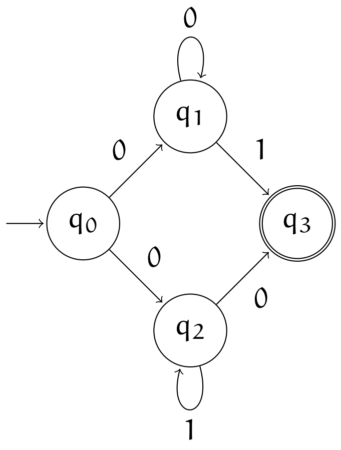

Figure 2: A rudimentary NFA.

[2]The power set of a set A is the set of all possible subsets of A. For example, $\mathcal{P}(\{a, b, c\})$ is

$$\{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}.$$

Example.

Figure 2 shows an example of an NFA. Notice that there are two 0-transitions from q_0, and no 1-transitions. This means $(q_0, 0)$ maps to the set $\{q_1, q_2\}$, and $(q_0, 1)$ maps to the empty set.

We can make similar observations about the other states, and construct a formal representation of this NFA.

1. The set of states is $Q = \{q_0, q_1, q_2, q_3\}$
2. The initial state is q_0.
3. The set of final states is $\{q_3\}$.
4. The input alphabet Σ is $\{0, 1\}$.
5. δ can be represented by this table:

δ	0	1
q_0	$\{q_1, q_2\}$	$\emptyset$
q_1	$\{q_1\}$	$\{q_3\}$
q_2	$\{q_3\}$	$\{q_2\}$
q_3	$\emptyset$	$\emptyset$

For the NFA in figure 2, given an input of 0001, you would start at q_0, then:

1. Read the first 0, and place fingers on both q_1 and q_2, since q_0 has 0-transitions to both of those states.
2. Read the second 0. The finger on q_1 stays in place, while the finger on q_2 goes to q_3.
3. Read the third 0. The finger on q_1 stays in place. As for the finger on q_3, since q_3 has no 0-transitions, you will remove this finger.
4. Read the 1. The finger on q_1 moves to q_3.
5. The input has been exhausted, and there is a branch on the final state q_3, so the string 0001 is accepted.

Now, let's cover the NFA$_\epsilon$. These are NFAs with ϵ-transitions. Knowing that ϵ is the empty string, these transitions are those which do not consume an input. Meaning, if you are on a state q and it has ϵ-transitions to states q_i (for some i)

2.3 Applications to String-Searching: KMP Algorithm

Automata have several applications. One of them is string-searching, which is the problem of finding occurrences of some pattern w inside of a string s.

There are several famous algorithms, most notably the Knuth-Morris-Pratt (KMP) algorithm and the Boyer-Moore algorithm. We'll only go over the former here.

The main idea of KMP is to ...

2.4 Applications to Games: Probabilistic Automata

some text.. relevance to markov chains

3 Regular Languages and Regular Expressions

DFAs can recognize a lot of different languages, but not all of them. The class of all languages that can be recognized by DFAs is known as the class of *regular languages*.

Let's first define some operations on languages:

Definition 3.1 (Operations on Languages).

- Given two languages A and B, the language which contains all strings in A and all strings B and no other strings is called the *union* of A and B, and is denoted $A \cup B$.
- Given two languages A and B, the *concatenation* of the two languages is the set of all strings of the form ab, where a is in A and b is in B. This language is denoted AB. Further, A^n is the concatenation of A with itself n times.
- Given a language A, the *Kleene closure* of A is denoted A^*. It is the union of all languages $A^0, A^1, A^2, \ldots$.

It would be good to present one example, at least.

Example.

Suppose $\Sigma = \{a, b\}$, A is the language $\{\epsilon, aa, aaaa, aaaaaa\}$, and B is the language $\{\epsilon, bbb, bbbbb\}$. Then,

- AB is the language

$$\{\epsilon, bbb, bbbbb, aa, aabbb, aabbbbb, aaaa, aaaabbb,$$
$$aaaabbbbb, aaaaaa, aaaaaabbb, aaaaaabbbbb\}$$

- $A \cup B$ is the language containing ϵ, aa, bbb, $aaaa$, $bbbbb$, $aaaaaa$, $bbbbbbbb$, $\ldots$
- A^* contains all strings with an even number of a's. As for B^*, it contains all strings with 0, 3, 5, 6, or 8 or more b's.[3]

Now to define regular languages according to these operations. The class of regular languages over some alphabet Σ uses a recursive definition; you start from a few base cases, and build up the rest by repeatedly applying different operations.

Definition 3.2 (Regular Language). The class of regular languages over an alphabet Σ is defined as follows:

- The empty language $\emptyset$ is a regular language.
- For each symbol a in Σ, the language $\{a\}$ is a regular language.
- If A is a regular language, then A^* is also a regular language.
- If A and B are regular languages, then $A \cup B$ and AB are regular languages.
- No other languages over Σ are regular.

Now, regular expressions are simply a notation to express a regular language. Union, concatenation, and closure are all that is needed (along with parentheses for grouping). Here's some examples of regular expressions over $\{a, b\}$ and what they mean.

[3]You can take this for granted, or read more here.

Example (Regular Expressions).

- $\mathbf{ab} + \mathbf{a}^*$ is a regular expression for the language $\{ab, \epsilon, a, aa, aaa, \ldots\}$.
- $(\mathbf{a} + \mathbf{b})^*$ is a regular expression for the language Σ^* of all strings composed of a's and b's.
- $\mathbf{b}^*\mathbf{ab}^*$ is a regular expression for the language of all strings containing exactly one a.

Regular expressions (or regex, for short) are extremely powerful and see heavy use in processing of strings. For example, your text editor or IDE uses them for syntax highlighting.[4]

3.1 Applications to Compilers: `Flex`

phases of compilation, etc..

[4]Usually regex does the job, but sometimes something more sophisticated is needed. For example, Visual Studio colors local variables and global variables differently, but regex by itself can't differentiate between the two. My hope is to cover some of the relevant techniques in the future.