Automata Theory

Abhijit Amrendra Kumar

August 2023

Chapter 1

Introduction

1.1 Abstract

Automata theory involves the study of systems of computations.

- (formal verification) What are the properties of a concrete algorithm given in a system of computation? What methods can be used to analyse such questions?
- (expressivity) What kind of calculations are possible in a system of computation? Can system A compute everything that is computable in system B?
- (computability) Is a problem solvable in given system of computation?

Some larger questions include

- Is there a most powerful system of mechanical computation ?
- Are there problems which cannot be mechanically solved ?

1.2 Course Syllabus

- **Finite State Automata**
  - Deterministic Finite Automata (DFA) and its uses.
  - DFA, NFA and Equivalence
  - Closure Properties and Decision Problems
  - Regular Expressions and Equivalence to DFA
  - Homomorphisms
  - DFA minization
  - Pumping Lemma
  - Myhill Nerode Theorem
- **Pushdown Automata and Context-Free Languages**
  - Phrase structured Grammars and Chomsky Hiearchy
  - Context Free Grammars (CFG), Uses of CFG, Normal forms
  - Push Down Automata (PDA)
  - Equivalence of CFG and PDA
- **Turing Machines and Computability**
  - Definition and Examples of Turing Machines
  - Robustness (Multihead TM, Nondeterministic TM)
  - Universal Turing Machines
  - Halting Problem and Undecidability
  - Variety of Unsolvable Problems
  - Many-to-One reductions
  - Rice Theorem

1.3 References

- Course Webpage
- Dexter Kozen, Automata and Computability. Springer, 1997.
- Hopcroft, Motwani, Ullman, Introduction to Automata Theory, Languages and Computation, PearsonEducation Asia, 2006.

Chapter 2

Words and Languages, Deterministic Finite Automata

2.1 Decision Problems and Automata

A decision problem is a problem which has `yes` or `no` as answer.

2.1.1 What is an Automaton ?

Consider a set of input instances `A` and a subset `B` of `A` which only contains instances for which the answer is `yes`. An **automaton** for (A, B) is a machine which can solve any given instance $x \in A$ of the decision problem: given $x \in A$, determine whether $x \in B$.

2.1.2 Problem Instances as Strings

To give input to an automaton, we will use strings of symbols.

- Alphabet Σ: A finite set of symbols/letters.
  - Ex. $\Sigma = \{0, 1\}$
  - We will use $a, b, c, ...$ to denote letters.
- Word over an alphabet Σ is a finite sequence of symbols from Σ.
  - Ex. 01001
  - We will use $x, y, z, ...$ to denote words.
- Σ^* denotes the set of all words over the alphabet Σ.
- ϵ denotes the empty word.
- $|x|$ denotes the length of word x.
  - Ex. $|101101| = 6$
  - $|\epsilon| = 0$
- Catenation of 2 words x, y is denoted by $x \cdot y$.
  - Ex. $ab \cdot abba = ababba$
  - $|x \cdot y| = |x| + |y|$
  - $x \cdot (y \cdot z) = (x \cdot y) \cdot z$
  - $\epsilon \cdot x = x \cdot \epsilon = x$
  - $x^{n+1} = (x^n) \cdot x$

2.2 Language

A **Language** over an alphabet Σ is any subset of Σ^*.

- We use $A, B, C, \ldots$ to denote languages
- We use $\#a(x)$ to denote the count of a in x
- Here are some examples
  - $A = \{a^p \mid p \text{ is a prime}\}$
  - $A = \{x \in \{0,1\}^* \mid \#0(x) = \#1(x)\}$
  - $A = \{x \in \{0,1\}^* \mid \exists i \in \mathbb{N},\ x \text{ is the binary representation of } 2^i\}$

2.2.1 Operations on Languages

Let A, B be languages over an alphabet Σ.

- $A \cup B = \{x \in \Sigma^* \mid x \in A \text{ or } x \in B\}$
- $A \cap B = \{x \in \Sigma^* \mid x \in A \text{ and } x \in B\}$
- $\sim A = \{x \in \Sigma^* \mid x \notin B\}$
- $A \cdot B = \{x \cdot y \mid x \in A \text{ and } y \in B\}$
  - Ex. $A = \{a, ab\}, B = \{b, ba\} \implies A \cdot B = \{ab, aba, abb, abba\}$
- **Language Recognition Problem**: Given an input word x, determine whether $x \in A$.
- All computational problems can be encoded into language recognition problems
- Each automaton defines a language

2.3 Deterministic Finite Automata

A DFA is given by $Q, \Sigma, \delta, q_0, F$, where

- Q is the set of states
- Σ is the set of alphabets
- $\delta : Q \times \Sigma \to Q$ is the transition function
- $q_0 \in Q$ is the initial state
- $F \subseteq Q$ is the set of final states

A DFA can also be represented by either a transition table, or via a transition diagram.

- A run of DFA A on $x = a_0, a_1, \ldots, a_{n-1}$ is a sequence of states $q_0, q_1, \ldots q_n$ s.t $q_i \xrightarrow{a_i} q_{i+1}$ for $0 \leq i < n$.
- For a given word x, DFAs have unique runs.
- A run is accepting if last state $q_n \in F$
- A word is accepted by an automaton if it's run is accepting

Language accepted by an automaton A can also be defined as

$$L(A) = \{u \in \Sigma^* | \text{ The run of A on u is accepting } \}$$

A language is called **regular** if it is accepted by some DFA.

2.3.1 Big Step Semantics

Given DFA with transition function $\delta : Q \times \Sigma \to Q$, we can define an extended transition function

$$\hat{\delta} : Q \times \Sigma^* \to Q$$

Here, $\hat{\delta}(q, x)$ gives the last state of the run of A on word x starting from state q. It can be defined inductively as follows

$$\begin{aligned} \hat{\delta}(q, \epsilon) &= q, \\ \hat{\delta}(q, ua) &= \delta(\hat{\delta}(q, u), a), \end{aligned}$$

A word x is accepted by an automaton A iff $\hat{\delta}(q_0, x) \in F$

Theorem:

$$\hat{\delta}(q, xy) = \hat{\delta}(\delta(\hat{q}, x), y)$$

Proof:

2.4 Constructions on Automata

Chapter 3

Regular Expressions

3.1 Automata to RegExp

Theorem:
For every NFA $A = (Q, \Sigma, \Delta, I, F)$, we can construct a language equivalent regular expression $reg(A)$

Proof:

Construction:
Define regular expression $\alpha_{p,q}^{X}$ for $X \subseteq Q, \quad p, q \in Q$. This set of words w such that A has a run on w from p to q, where all the intermediate states are from X.

Examples:

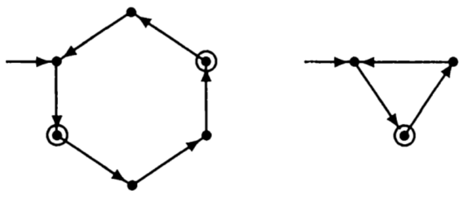
$$\alpha_{p,q}^{r} = 1 \cdot 0 \qquad\qquad \alpha_{p,r}^{p,q} = 0^* \cdot 1 \cdot 0$$

Let's try to compute $\alpha_{p,p}^{p,q,r}$.

$\alpha_{p,r}^{X}$ can be recursively

Chapter 4

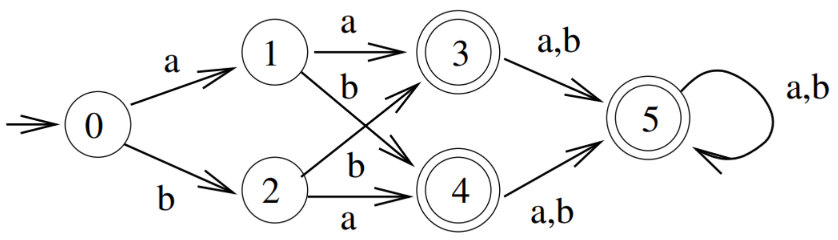
Minimizing DFA



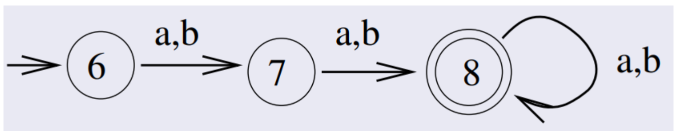
There can be multiple DFAs recognizing the same language. Hence the question arises as to if there exists a minimum DFA which can recognize the same language, and if so, how to find it.

We already know that ϵ-NFA to DFA conversion often gives rise to large automaton which are not guaranteed to be optimal in size. The same is true when converting regular expressions to DFA.

Let's consider some examples to notice some patterns.



Here's the minimal DFA for the language defined by the above DFA.



To minimise a DFA, let's first define the following:

Equivalent States in a DFA:
Two states p, q in an DFA are said to be equivalent states (denoted by $p \approx q$) if

$$p \approx q \overset{\text{def}}{=} \forall x \in \Sigma^* (\hat{\delta}(p, x) \in F \iff \hat{\delta}(q, x) \in F)$$

The proposition $\approx$ is an equivalence relation

- $\forall p,\ p \approx p$
- $\forall p, q,\ p \approx q \implies q \approx p$
- $\forall p, q, r,\ p \approx q \wedge q \approx r \implies p \approx r$