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 $A, B, C, ...$ 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 $x = a_0, a_1, ..., a_{n-1}$ is a sequence of states $q_0, q_1, ...q_n$ s.t $q_i \xrightarrow{a_i} q_{i+1}$ for $0 \leqslant i < n$.
- 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

$$\hat{\delta}(q, \epsilon) = q,$$
$$\hat{\delta}(q, ua) = \delta(\hat{\delta}(q, u), a),$$

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 $\alpha^X_{p,q}$ 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:

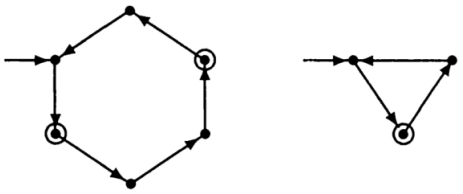
$$\alpha^r_{p,q} = 1 \cdot 0 \qquad\qquad \alpha^{p,q}_{p,r} = 0^* \cdot 1 \cdot 0$$

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

$\alpha^X_{p,r}$ can be recursively

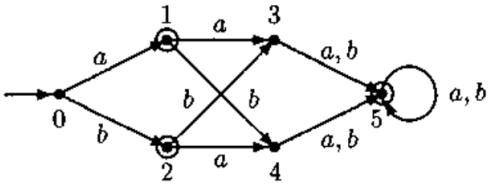
Chapter 4

Minimizing DFA

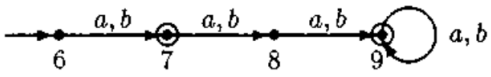


There can be multiple DFAs recognizing the same language. Hence the question arises as to if there exists a minimum DFA which can recognize the same language, and if so, how to find it.
We already know that ϵ-NFA to DFA conversion often gives rise to large automaton which are not guaranteed to be optimal 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.



The minimization procedure consists of 2 stages:

- Get rid of inaccessible states; that is, states q for which there exists no string $x \in \Sigma^*$ such that $\hat{\delta}(s, x) = q$
- Collapse equivalent states

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

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

The proposition $\approx$ is an equivalence relation

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

It partitions Q into equivalence classes. Let $[p]$ denote the equivalence classes of p.

Quotient Automaton

Given DFA $M = (Q, \Sigma, \delta, [q_0], F)$ and $\approx$ as before, the Quotient automaton is $M/\approx \stackrel{\text{def}}{=} (Q', \Sigma, \delta', [q_0], F')$, where

$$\begin{aligned} Q' &= \{[p] \mid p \in Q\} \\ \delta'([p], a) &= [\delta(p, a)] \\ F' &= \{[f] \mid f \in F\} \end{aligned}$$

Lemma: $p \approx q \implies \forall a \in \Sigma.\quad \delta(p,a) \approx \delta(q,a)$
Proof:
Suppose $p \approx q$. Let $a \in \Sigma$, and $y \in \Sigma^*$

$$\begin{aligned} \hat{\delta}(\delta(p,a), y) \in F &\iff \hat{\delta}(p, ay) \in F \\ &\iff \hat{\delta}(q, ay) \in F \\ &\iff \hat{\delta}(\delta(q,a), y) \in F \end{aligned}$$

Since y was arbitrary, $\delta(p,a) \approx \delta(q,a)$ by definition of $\approx$.

Lemma: $p \in F \Leftrightarrow [p] \in F'$
Proof:
LHS implies RHS is evident by definition. For proving that RHS implies LHS, we need to show that if $p \approx q$ and $p \in F$, then $q \in F$. In other words, every equivalence class is either a subset of F, or disjoint from F. This follows by taking $x = \epsilon$ in the definition of $p \approx q$.

Lemma: $\forall\ x \in \Sigma^*, \hat{\delta}'([p], x) = [\hat{\delta}(p, x)]$.
Proof: By induction on $|x|$
Basis: For $x = \epsilon$,

$$\begin{aligned} \hat{\delta}'([p], \epsilon) &= [p] \\ &= [\hat{\delta}(p, \epsilon)] \end{aligned}$$

Induction step: Assume $\hat{\delta}'([p], x) = [\hat{\delta}(p, x)]$ and let $a \in \Sigma$

$$\begin{aligned} \hat{\delta}'([p], xa) &= \delta'(\hat{\delta}'([p], x), a) \\ &= \delta'([\hat{\delta}(p, x)], a) \\ &= [\delta(\hat{\delta}(p, x), a)] \\ &= [\hat{\delta}(p, xa)] \end{aligned}$$

Theorem: $L(M/\approx) = L(M)$
Proof:

$$\begin{aligned} x \in L(M/\approx) &\iff \hat{\delta}'([q_0], x) \in F' \\ &\iff [\hat{\delta}(q_0, x)] \in F' \\ &\iff \hat{\delta}(q_0, x) \in F \\ &\iff x \in L(M) \end{aligned}$$

$M/\approx$ cannot be collapsed further

It is conceivable that after doing the quotient construction once, we might be able to collapse even further by doing it again. It turns out that once is enough. To see this, let's do the quotient construction a second time.
Define

$$[p] \sim [q] \stackrel{\text{def}}{\iff} \forall\ x \in \Sigma^* (\hat{\delta}'([p], x) \in F' \iff \hat{\delta}'([q], x) \in F')$$

This is exactly the same definition as $\approx$ above, only that it is applied to $M/\approx$. Now:

$$\begin{aligned}
[p] \sim [q] &\implies \forall\ x \in \Sigma^*(\hat{\delta}'([p], x) \in F' \iff \hat{\delta}'([q], x) \in F') \\
&\implies \forall\ x \in \Sigma^*([\hat{\delta}(p, x)] \in F' \iff [\hat{\delta}(q, x)] \in F') \\
&\implies \forall\ x \in \Sigma^*(\hat{\delta}(p, x) \in F \iff \hat{\delta}(q, x) \in F) \\
&\implies p \approx q \\
&\implies [p] = [q]
\end{aligned}$$

Minimization algorithm

- Make pairs table with $(p, q) \in Q \times Q$ and $p \leqslant q$.
- Mark (p, q) if $p \in F \wedge q \notin F$ or vice versa.
- Repeat following steps until no change occurs.
  - Pick each unmarked state (p, q).
  - If $(\delta(p, a), \delta(q, a))$ is marked for some $a \in \Sigma$ then mark (p, q).
- For each pair, $p \approx q$ iff (p, q) is unmarked.

Termination: In each pass at least one new pair must get marked.

Theorem: (p, q) is marked iff $\exists x \in \Sigma^*$, $\hat{\delta}(p, x) \in F \wedge \hat{\delta}(q, x) \notin F$ or vice versa iff $p \not\approx q$
Proof:
Base

A nice way to look at the automaton is as a finite automaton itself. Let

$$Q = \{\{p, q\} | p, q \in Q, p \neq q\}$$

Define a non-deterministic transition function $\Delta : Q \to 2^Q$ on Q as follows:

$$\Delta(\{p, q\}, a) = \{\{p', q'\} | p = \delta(p', a), q = \delta(q', a)\}$$

Define a set of start states $S \subseteq Q$ as follows:

$$S = \{\{p, q\} | p \in F, q \notin F\}$$

Step 2 of the algorithm marks the elements in S, and step 3 marks the pairs in $\Delta(\{p, q\}, a)$ when $\{p, q\}$ is marked for any $a \in \Sigma$. In these terms, the above theorem says that $p \not\approx q$ if $\{p, q\}$ is accessible in this automaton.

Chapter 5

Myhill-Nerode Relations

5.1 Introduction

Two DFAs $M = (Q_M, \Sigma, \delta_M, s_M, F_M), N = (Q_N, \Sigma, \delta_N, s_N, F_N)$ are said to be **isomorphic** if there is a bijection $f : Q_M \to Q_N$ such that

- $f(s_M) = s_N$
- $f(\delta_M(p, a)) = \delta_N(f(p), a) \forall\ p \in Q_M, a \in \Sigma$
- $p \in F_M \iff f(p) \in F_N$

In other words, they are essentially the same automaton, but with different names for their states.

5.2 Definition

Let $\equiv$ be an equivalence relation over Σ^*.

- $\equiv$ is right congruent if $\forall x, y \in \Sigma^*, a \in \Sigma$ we have

$$x \equiv y \quad \Rightarrow \quad x \cdot a \equiv y \cdot a$$

- Let $R \subseteq \Sigma^*$ (not necessarily regular). $\equiv$ refines R if $\forall x, y \in \Sigma^*$

$$x \equiv y \quad \Rightarrow \quad (x \in R \Leftrightarrow y \in R)$$

- $\equiv$ is of finite index if $\equiv$ partitions the Σ^* into only finitely many equivalence classes.
- Equivalence relation $\equiv$ is called Myhill-Nerode Relation refining R if it satisfies all the three properties above.
- $\equiv$ is called Weak Myhill Nerode relation Refining R if it satisifies the first two properties.

5.3 Machine Equivalence

Given a DFA $A = (Q, \Sigma, \delta, q_0, F)$, define the induced equivalence $\equiv_A$ over Σ^* as follows:

$$x \equiv_A y \stackrel{\text{def}}{=} \hat{\delta}(q_0, x) = \hat{\delta}(q_0, y)$$

Proposition $\equiv_A$ is a Myhill-Nerode relation refinning $L(A)$.
Proof Method: Check the following

- $\equiv_A$ is an equivalence relation for Σ^*
- $x \equiv_A y \implies \forall\ a, xa \equiv_A ya$
- $x \equiv_A y \implies (x \in L(A) \iff y \in L(A))$
- $\equiv_A$ is of finite index

5.4 From Equivalence to DFA

Let $\equiv_A$ be Myhill-Nerode refining R. Define DFA $A_\equiv \stackrel{\text{def}}{=} (Q, \Sigma, \delta, S, F)$ as follows:

$$\begin{aligned} Q &\stackrel{\text{def}}{=} \{[x] \mid x \in \Sigma^*\} \\ q_0 &\stackrel{\text{def}}{=} [\epsilon] \\ F &\stackrel{\text{def}}{=} \{[x] \mid x \in R\} \\ \delta([x], a) &\stackrel{\text{def}}{=} [xa] \end{aligned}$$

Theorem 5.4.1. $L(A_\equiv) = R$

Proof.

$$\begin{aligned} x \in L(A_\equiv) &\iff \hat{\delta}([\epsilon], x) \in F \\ &\iff [\epsilon x] \in F \\ &\iff x \in R \end{aligned}$$

□

Lemma 5.4.2. $\hat{\delta}([x], y) = [xy]$

Proof. Induction on y
Base step

$$\hat{\delta}([x], \epsilon) = [x] = [x\epsilon]$$

Induction step
Assuming the lemma holds for y,

$$\begin{aligned} \hat{\delta}([x], ya) &= \delta(\hat{\delta}([x], y), a) \\ &= \delta([xy], a) \\ &= [xya] \end{aligned}$$

□

5.5 Correspondence

The $\equiv_A$ and $A_\equiv$ are inverses of each other.

Theorem 5.5.1. $\equiv_{A_\equiv} = \equiv$

Theorem 5.5.2. *If A is an automaton without unreachable states, then $A_\equiv$ is isomorphic to A*

5.6 Refining Equivalences