# Theory Of Computation

Muyao Xiao

Nov. 2024

**Abstract**

The note is taken by studying 18.404J/6.5400 *Theory of Computation* course by professor Michael Sipser of MIT. The course material can be downloaded in MIT OpenCourseWare. Meanwhile, most contents in this note will also be derived from his book *Introduction to the Theory of Computation, third edition.*

The course is divided into 2 parts, computational theory and complexity theory. Computational theory is developed during 1930s - 1950s. It concerns about what is computable. This note will be focused on the first part.

**Example.** Program verification, mathematical truth

**Example** (Models of Computation)**.** Finite automata, Turing machines, $\cdots$

# Contents

# Chapter 1

# Introduction, Finite Automata, Regular Expressions

The theory of computation begins with a question: What is a computer. The real computer is too complicated to understand, to start with, we use an idealized computer called **computational model**.

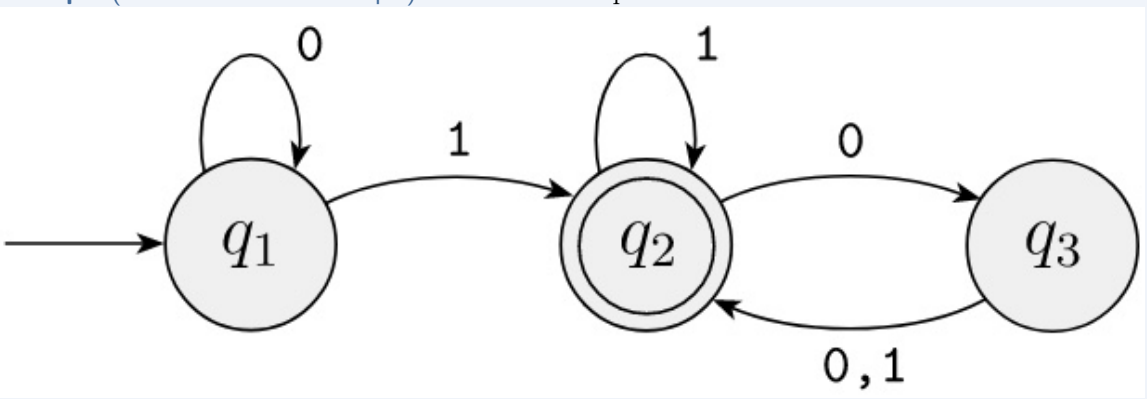
The simplest model among them is **finite state machine** or **finite automaton**.

## 1.1 Finite Automata

Finite automata are good models for computers with an extremely limited amount of memory.

Finite automata and their probabilistic counterpart **Markov chains** are useful tools when we're attempting to recognize patterns in data. Markov chains have even been used to model and predict price changes in financial markets.

**Example** (Finite Automata Example)**.** Here's an example of finite automata:




- The figure is called **state diagram** of $M_1$.
- Three **states**: $q_1$, $q_2$ and $q_3$.
- **Start state**: $q_1$.
- **Accept state**: $q_2$.
- The arrows going from one state to another are called **transitions**.

When the automaton receives an input string such as `1101`, it processes that string and produces an output. The output is either **accept** or **reject**:

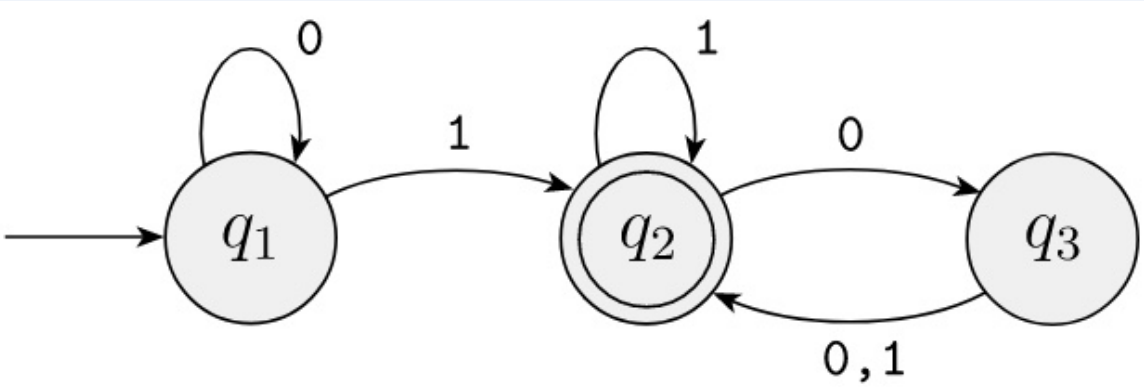
1. Start in state $q_1$
2. Read `1`, follow transition from $q_1$ to $q_2$

3. Read 1, follow transition from $q_2$ to $q_2$
4. Read 0, follow transition from $q_2$ to $q_3$
5. Read 1, follow transition from $q_3$ to $q_2$
6. *Accept* because $M_1$ is in an accept state $q_2$ at the end of the input

**Definition 1.1.1** (Formal Definition of A Finite Automaton)**.** A **finite automaton** is a 5-tuple $(Q, \Sigma, \sigma, q_0, F)$, where

1. $Q$ is a finite set called **state**
2. $\Sigma$ is a finite set called the **alphabet**
3. $\sigma : Q \times \Sigma \Rightarrow Q$ is the **transition function**
4. $q_0 \in Q$ is the **start state**
5. $F \subseteq Q$ is the **set of accept state**

**Example** (Revisit Finite Automata Example)**.** Let's revisit the finite automata example $M_1$ and see from the formal definition perspective:




We can describe $M_1$ formally by writing $M_1 = (Q, \Sigma, \sigma, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3\}$
2. $\Sigma = \{0, 1\}$
3. $\sigma$ is described as

| | 0 | 1 |
|---|---|---|
| $q_1$ | $q_1$ | $q_2$ |
| $q_2$ | $q_3$ | $q_2$ |
| $q_3$ | $q_2$ | $q_2$ |

4. $q_1$ is the start state
5. $F = \{q_2\}$

If A is the set of all strings that machine $M$ accepts, we say that A is the **language of machine M** and write $L(M) = A$. We say that **M recognizes A** or that **M accepts A**. Here because *accept* has different meaning, we use *recognize* for the language.

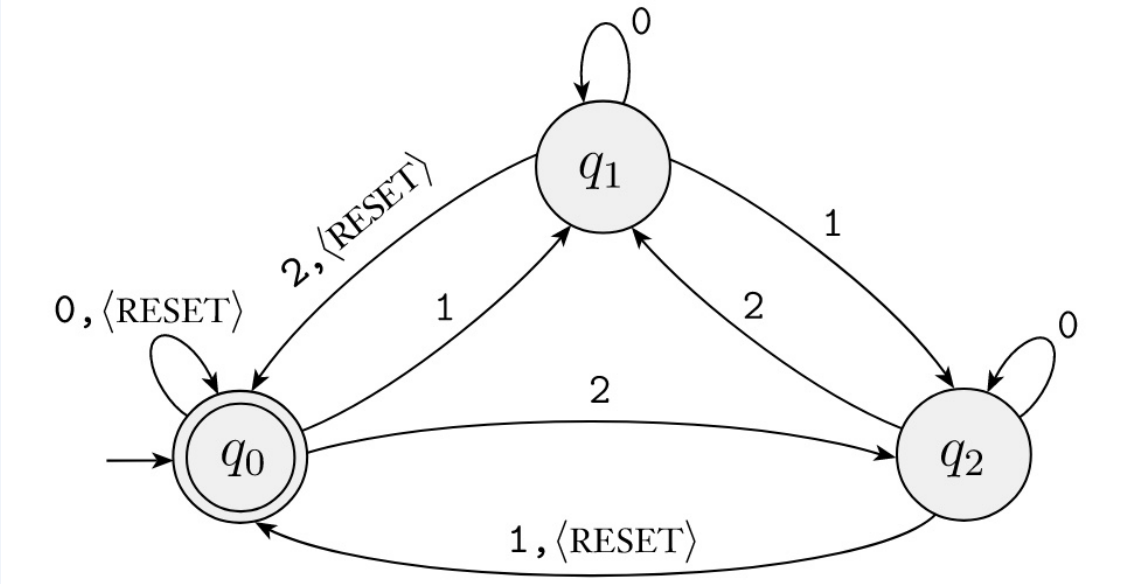
**Remark.** A machine may accept several strings, but it always recognizes only one language. If the machine accepts no strings, it still recognizes one language – namely, the empty language $\emptyset$.

**Example** (Revisit Finite Automata Example: Language)**.** In our example, the language set A can be represented as:

A = $\{\omega | \omega$ contains at least one `1` and an even number of `0`s follow the last `1`$\}$.

Then $L(M_1) = A$, or equivalently, $M_1$ recognizes $A$.

**Example.** When describing such a machine:




The alphabet $\Sigma = \{1, 2, 3, < RESET >\}$, we treat $< RESET >$ as a single symbol.

The machine keeps a running count of the sum of the numerical input symbols it reads, modulo 3. Every time it receives $< RESET >$ symbol, it resets the count to 0. It accepts if the sum is 0 modulo 3.

## 1.2 Formal Definition of Computation

Let $M = (Q, \Sigma, \sigma, q_0, F)$ be a FA and let $\omega = \omega_1\omega_2 \cdots \omega_n$ be a string where each $\omega_i$ is a member of alphabet $\Sigma$. Then M **accepts** $\omega$ if a sequence of state $r_0, r_1, \cdots, r_n$ in $Q$ exists with three conditions:

1. $r_0 = q_0$ (machine starts at initial state)
2. $\sigma(r_i, \omega_{i+1}) = r_{i+1}$ (machine goes form state to state following the transition function)
3. $r_n \in F$ (machine accepts its input if it ends up in an accept state)

We say that M recognizes language A if $A = \{\omega | Maccepts\omega\}$

**Note.** A is the language, $\omega$ is the accepted string. A is the set of all instances of $\omega$.

We say a machine "accepts" a string, and a machine "recognizes" a language.

**Definition 1.2.1** (Regular Language)**.** A language is called a **regular language** if some finite automaton recognizes it.

**Example.** Let B = $\{ \omega \mid \omega$ has even number of 1s $\}$

B is a regular language.

**Example.** Let C = $\{ \omega \mid \omega$ has equal numbers of 0s and 1s $\}$

C is <u>not</u> a regular language.

## 1.3 Regular Expressions

### 1.3.1 Regular Operations

**Definition 1.3.1.** Let A and B be languages, we define the regular operations **union** , **concatenation** , and **start** as follows:

- Union: $A \cup B = \{x | x \in A || x \in B\}$
- Concatenation: $A \circ B = \{xy | x \in A \& y \in B\}$
- Star: $A^* = \{x_1 x_2 \cdots x_k | k \geq 0 \& x_i \in A\}$

Notice that $\epsilon$(empty language) always belongs to $A*$.

**Example.** $\Sigma^*1$ is the language end with 1

**Remark.** Show finite automata equivalent to regular expressions.

### 1.3.2 Closure Properties

**Theorem 1.3.1.** The class of regular language is closed under the union operation.
In other words, if $A_1$ and $A_2$ are regular languages, so is $A_1 \cup A_2$.

**Proof.** Let $M_1 = \{Q_1, \Sigma, \sigma_1, q_1, F_1\}$ recognize $A_1$.
Let $M_2 = \{Q_2, \Sigma, \sigma_2, q_2, F_2\}$ recognize $A_2$. (assuming in the same alphabet to make the proof simple)
Construct $M = (Q, \Sigma, \sigma, q_0, F)$ recognizing $A_1 \cup A_2$.
M should accept input w if either $M_1$ or $M_2$ accepts w.
Component of M:

- $Q = Q_1 \times Q_2$
- $q_0 = (q_1, q_2)$
- $\sigma((q, r), a) = (\sigma_1(q, a), \sigma(r, a))$
- $F = (F_1 \times Q_2) \cup (Q_1 \times F2)$
- not $F = \cancel{F_1 \times F2}$ (this gives intersection!)

■

**Example** (What is close?). Positive integers close under addition but not close under subtraction.

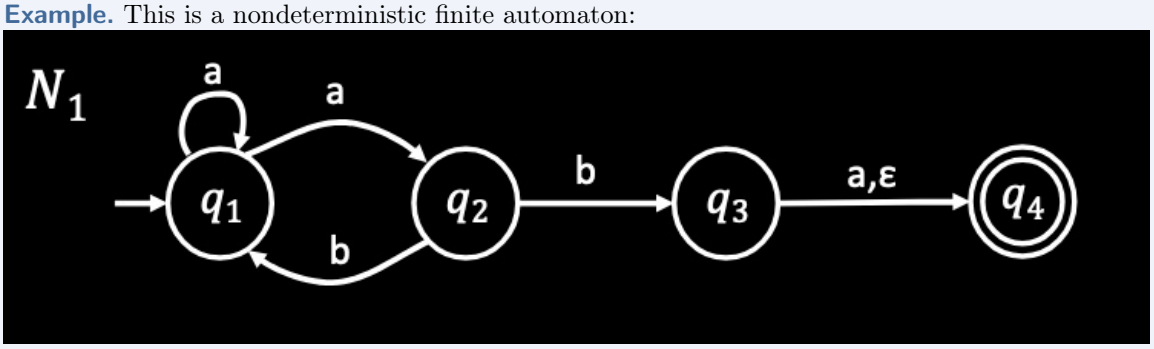
**Theorem 1.3.2.** The class of regular language is closed under the concatenation operation.
In other words, if $A_1$ and $A_2$ are regular languages then so is $A_1 \circ A_2$.

# Chapter 2

# Nondeterminism, Closure Properties, Regular Expressions

## 2.1 Nondeterminism

**Example.** This is a nondeterministic finite automaton:




What's the difference with what we saw in the last lecture:

- in $q_1$, when accepting an $a$, you can either stay in $q_1$ or go to $q_2$
- in $q_1$, if get $b$ then there's no where to go
- ...

Example inputs

- ab (accept)
- aa (reject)

New features of nondeterminism:

- multiple paths possible (0, 1 or many at each step)
- $\epsilon$-transition is a "free" move without reading input
- Accept input if some path leads to accept state (acceptance overrules rejection) (if one of possible ways to go accepts, then accepts)

Nondeterminism doesn't correspond to a physical machine we can build, however it is useful mathematically.

## 2.2 NFA

**Definition 2.2.1.** A **nondeterministic finite automaton** is a 5-tuple $Q, \Sigma, \sigma, q_0, F$, where

1. Q is a finite set of states
2. $\Sigma$ is a finite alphabet
3. $\sigma : Q \times \Sigma_\epsilon \Rightarrow \mathsf{P}(Q)$ is the transition function
4. $q_0 \in Q$ is the start state
5. $F \subseteq Q$ is the set of accept states

In which, $\Sigma_\epsilon$ is a shorthand of $\Sigma \cup \{\epsilon\}$. P(Q) means the power set of Q which can be represented as $P(Q) = R | R \subseteq Q$, which is the set which contains all the subset of Q.

**Example.** Check the