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**.

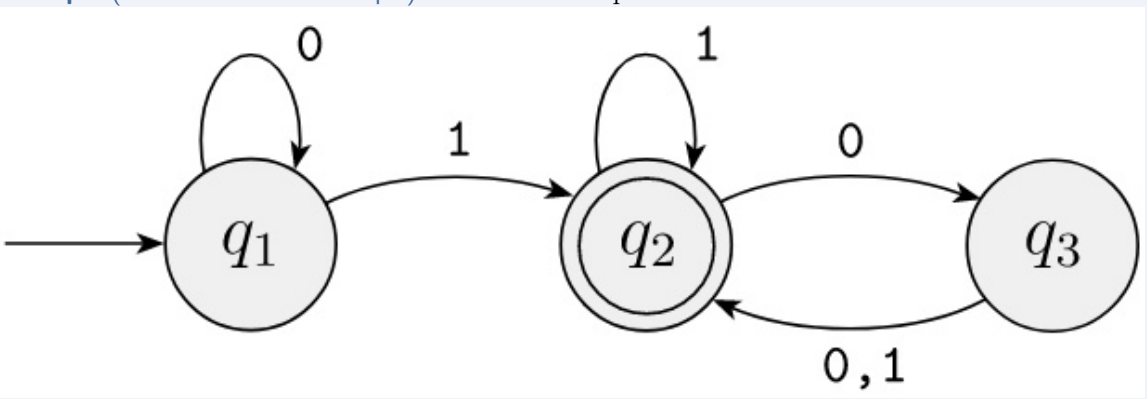
The simplest model among them is **finite state machine** or **finite automaton**.

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**:

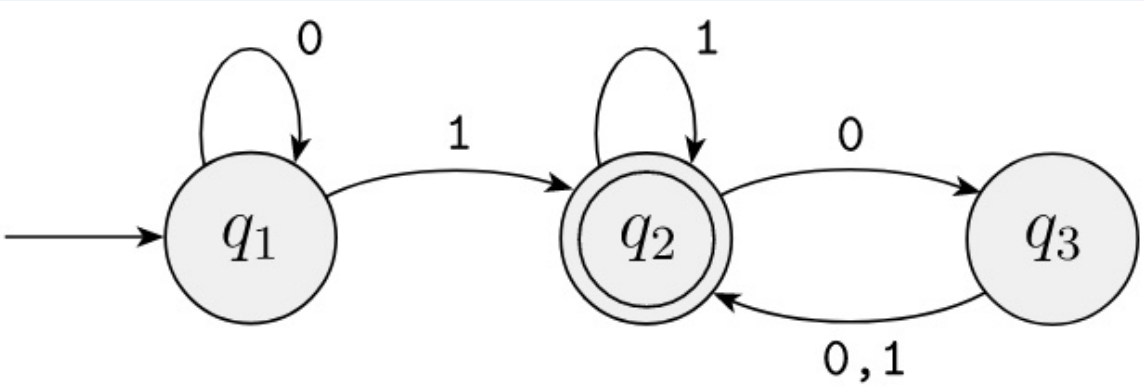
1. Start in state q_1
2. Read `1`, follow transition from q_1 to q_2

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. Σ 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. σ 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.