1 Introduction

1.1 finite automaton

finite automaton is used in :

- text processing
- compilers
- hardware design

1.2 context-free grammar

context-free grammar is used in :

- programming languages
- artificial intelligence

1.3 set

a **set** is a group of objects represented as a unit .
The objects in a set are called its **elements** or **members** .
a set with one member is called the **empty set** .
a set with one member is sometimes called a **singleton set** .
a set with two members is called an **unordered pair** .

1.4 Sequence and Tuples

a **sequence** of objects is a list of these objects in some order .
we usually designate a sequence by writing the list within parentheses . For example : $(7, 21, 57)$
The order doesn't matter in a set, but in a sequence it does .
repetition does matter in sequence, but it doesn't matter in a set .
Finite sequences often are called **tuples** .
a sequence with k elements is a **k-tuple** .
a 2-tuple is also called an **ordered pair** .

1.5 alphabet

alphabet is a finite non-empty set of objects called symbols .

1.6 string over an alphabet

a string over an alphabet is a finite sequence of symbols from that alphabet .

1.7 language

a language is a set of strings .

1.8 Summary

Alphabet a finite non-empty set of objects called symbols

Empty set The set with no members

Empty string The string of length zero

k-tuple ordered list of k objects

Language a set of strings

Member an object in a set

sequence ordered list of objects

ordered pair sequence of two elements

unordered pair a set with two membes

set a group of objects

string a finite list of symbols from an alphabet

symbol a member of an alphabet

2 Regular Languages

2.1 Formal Definition Of a Finite Automaton

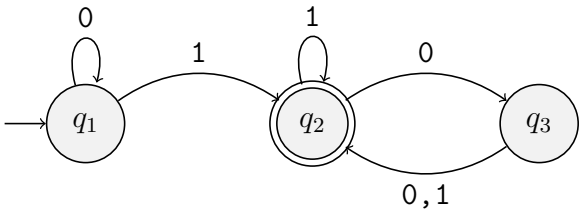
The formal definition says that a finite automaton is a list of these five objects :

1. set of states
2. input alphabet
3. rules for moving
4. start state
5. accept states

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

1. Q is a finite set called the **states**
2. Σ is a finite set called **alphabets**
3. $\delta : Q \times \Sigma \to Q$ is the **transition function**
4. $q_0 \in Q$ is the **start state**
5. $F \subseteq Q$ is the set of **accept states**

2.2 Example



We can describe the finite automata M formally writing $M = (Q, \Sigma, \delta, q_1, F)$, where :

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

Transition Table

	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. $F = \{q_2\}$

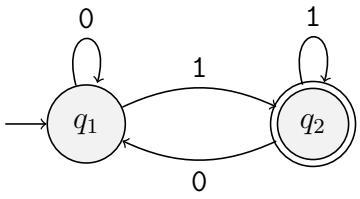
if A is the set of all strings that machine M accepts, we say that A is the **language of machine M** and write $L(M) = A$. We also say that **M recognizes A** of **M accepts A** .

a machine may accept several string, but it always recognizes only one language .

if the machine accepts no strings, it still recognized one languages, the empty language $\emptyset$.

2.3 Example

You can see the state diagram of the two-state finite automaton M :



in the formal description , $M = (Q, \Sigma, \delta, q_1, F)$, where :

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

Transition Table

	0	1
q_1	q_1	q_2
q_2	q_1	q_2

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

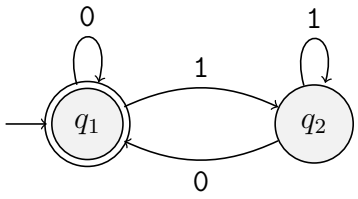
- after trying a few examples, you would see that M accepts all strings that end in 1 . Thus

$$L(M) = \{w \mid w \ ends \ in \ a \ 1\}$$

- Note that the machine accepts all strings that leave it in an accept state when it has finished reading .

2.4 Example

in the following Machine we just **change the position of accepting state** .



- the language of Machine M is :

$$L(M) = \{w \mid w \text{ is the empty string } \varepsilon \text{ or ends in } 0\}$$

2.5 Formal Definition Of Computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \dots w_n$ be a string where each w_i is a member of the alphabet Σ . Then M **accepts** w is a sequence of states $r_0, r_1, \dots, r_n$ in Q exists with three conditions :

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, 1, 2, \dots, n-1$
3. $r_n \in F$

Condition 1 says that the machine starts in the start state .

Condition 2 says that the machine goes from state to state according to the transition function

Condition 3 says that the machine accepts its input if it ends up in an accept state

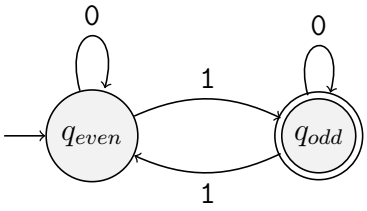
- we say that M **recognize language** A if $A = \{w \mid M \text{ accepts } w\}$

- a language is called a **regular language** if some finite automaton recognizes it

2.6 Example

construct a finite automaton to recognize all strings with an odd number of 1s ?
The alphabet is : $\Sigma = \{0, 1\}$

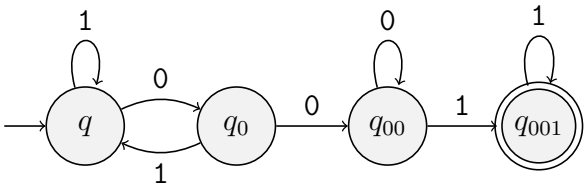
Answer :



2.7 Example

design a finite automaton to recognize all strings that contain the substring 001 ?

Answer :



2.8 NonDeterminism

when the machine is in a given state and reads the next input symbol , we know that the next state will be — it is determined . We call this **deterministic** computation .
in a **nondeterministic** machine , several choices may exist for the next state at any point .

2.9 DFA vs. NFA

The difference between a deterministic finite automaton, abbreviatedDFA, and a nondeterministic finite automaton, abbreviatedNFA :

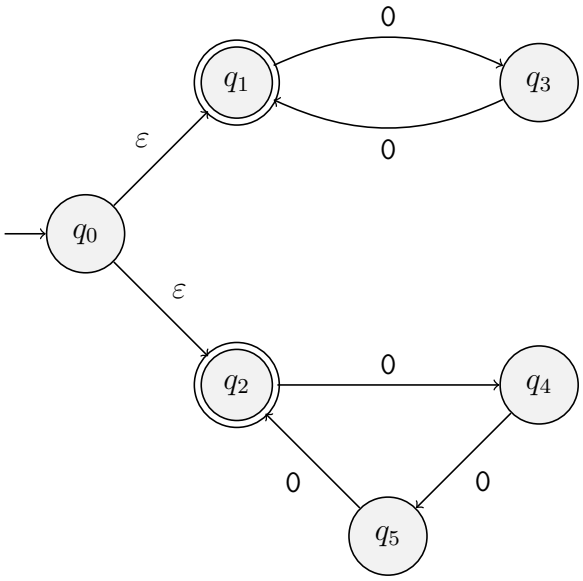
- every state of aDFAalways has exactly one exiting transition arrow for each symbol in the alphabet. In an NFA , a state may have zero , one , or many exiting arrows for each alphabet symbol.
- in a DFA , labels on the transition arrows are symbols from the alphabet. an NFA may have arrows labeled with members of the alphabet or ε . Zero , one , or many arrows may exit from each state with the label ε .

2.10 How does an NFA compute ?

Suppose that we are running an NFA on an input string and come to a state with multiple ways to proceed. After reading that symbol, the machine splits into multiple copies of itself and follows all the possibilities in parallel. Each copy of the machine takes one of the possible ways to proceed and continues as before. If there are subsequent choices, the machine splits again. If the next input symbol doesn't appear on any of the arrows exiting the state occupied by a copy of the machine, that copy of the machine dies, along with the branch of the computation associated with it. Finally, if any one of these copies of the machine is in an accept state at the end of the input, the NFA accepts the input string. If a state with an ε symbol on an exiting arrow is encountered, something similar happens. Nondeterminism may be viewed as a kind of parallel computation wherein multiple independent "processes" or "threads" can be running concurrently.

2.11 Example

design a NFA accepts all strings of the form 0^k where k is multiple of 2 or 3 .



2.12 Formal Definition of a Nondeterministic Finite Automaton

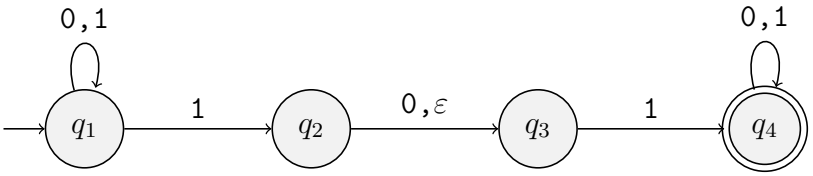
The formal definition of a nondeterministic finite automaton is similar to that of a deterministic finite automaton . Both have states , an input alphabet , a transition function, a start state, and a collection of accept states. However, they differ in one essential way: in the type of transition function. In a DFA , the transition function takes a state and an input symbol and produces the next state. In an NFA , the transition function takes a state and an input symbol or the empty string and produces the set of possible next states. For any set Q we write $P(Q)$ to be the collection of all subsets of Q. Here $P(Q)$ is called the power set of Q. For any alphabet Σ we write Σ_ε to be $\Sigma \cup \{\varepsilon\}$.

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

1. Q is a finite set of states
2. Σ is a finite alphabet
3. $\delta : Q \times \Sigma_\varepsilon \to P(Q)$ is the transition function
4. $q_0 \in Q$ is the start state
5. $F \subseteq Q$ is the set of accept states

2.13 Example

Suppose the following NFA :



The formal description of this NFA is $(Q, \Sigma, \delta, q_1, F)$, where :

1. $Q = \{q_1, q_2, q_3, q_4\}$
2. $\Sigma = \{0, 1\}$
3. δ is described as :

Transition Table

	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

4. q_1 is the start state
5. $F = \{q_4\}$

2.14 Theorem

- every nondeterministic finite automaton has an equivalent deterministic finite automaton .

- above Theorm state that every NFA can be converted into an equivalent DFA . Thus nondeterministic finite automata give an alternative way of characterizing the regular languages .

- the result of above facts is $\rightarrow$ a language is regular if and only if some nondeterministic finite automaton recognizes it .

2.15 Regular Expressions

the language consisting of all strings starting with a 0 or 1 followed by any number of 0s .

$$(0 \cup 1)0^*$$

the language consisting of all possible strings of 0s and 1s .

$$(0 \cup 1)^*$$

suppose $\Sigma = \{0, 1\}$, so the language contains all strings that end in a 1 .

$$\Sigma^* 1$$

suppose $\Sigma = \{0, 1\}$, so the language consists of all strings that start with 0 or ends with 1 .

$$(0\Sigma^*) \cup (\Sigma^* 1)$$

2.16 Formal Definition of a Regular Expression

Say that R is a **regular expression** if R is :

- any symbol in alphabet Σ
- ε
- $\emptyset$
- $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions
- $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions
- (R_1^*), where R_1 is a regular expression

Note :

- R^+ be shorthand for RR^*
- $R^+ \cup \varepsilon = R^*$

2.17 Examples

assume the alphabet $\Sigma = \{0, 1\}$:

1. $0^*10^* = \{w|w$ contains a single 1 $\}$
2. $\Sigma^*1\Sigma^* = \{w|w$ has at least one 1 $\}$
3. $\Sigma^*001\Sigma^* = \{w|w$ contains the string 001 as a substring $\}$
4. $1^*(01^+)^* = \{w|$ every 0 in w is followed by at least one 1 $\}$
5. $(\Sigma\Sigma)^* = \{w|w$ is a string of even length $\}$
6. $(\Sigma\Sigma\Sigma)^* = \{w|$ the length of w is a multiple of 3 $\}$
7. $01 \cup 10 = \{01, 10\}$
8. $0\Sigma^* \cup 1\Sigma^* \cup 0 \cup 1 = \{w|w$ starts and ends with the same symbol $\}$
9. $(0 \cup \varepsilon)1^* = 01^* \cup 1^*$
10. $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{ \varepsilon , 0 , 1 , 01 \}$
11. $1^*\emptyset = \emptyset$
12. $\emptyset^* = \{\varepsilon\}$

- The **length** of a string is the number of symbols that it contains .

2.18 Equivalence With Finite Automata

any regular expression can be converted into a finite automaton that recognizes the language it describes,and vice versa .

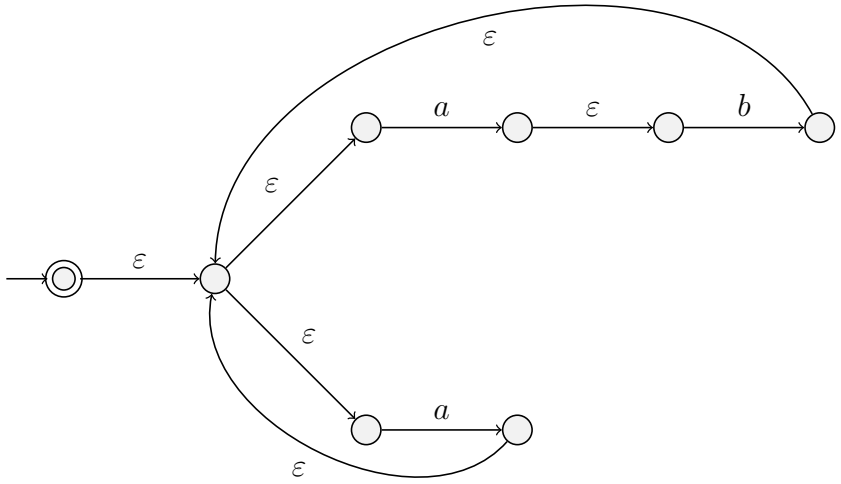
- a language is regular if and only if some regular expression describes it .

2.19 Lemma

- if a language is describes by a regular expression, then it is regular .

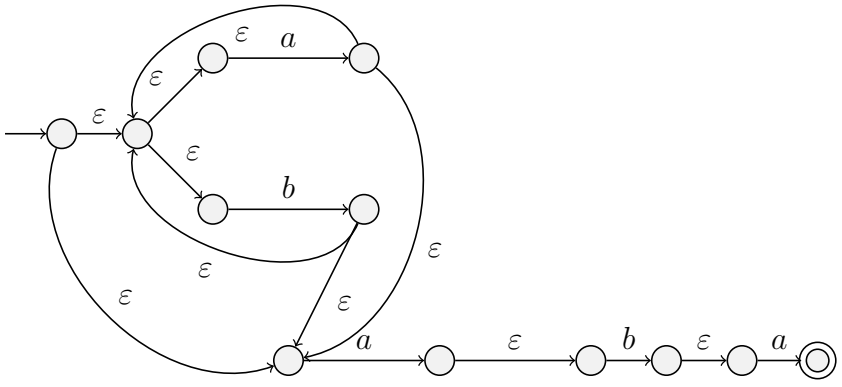
2.20 Example

Building an NFA from regular expressions $(ab \cup a)^*$



2.21 Example

Building an NFA from regular expressions $(a \cup a)^*aba$



2.22 Lemma

- if a language is regular, then it is described by a regular expression .

3 Context-Free Grammars

context-free grammars can describe certain features that have a recursive structure .
Context-free grammars were first used in the study of human languages .
an other important application of context-free grammars occurs in the application and compilation of programming languages .

3.1 Context-Free Grammars Terms

The following is an example of a context-free grammars :

$$A \to 0A1$$
$$A \to B$$
$$B \to \#$$

- a grammar consists of a collection of **substitution rules**, also called **productions**
- each symboled is called a **variable**
- The string consists of variables and **terminals**
- the variable often are represented by capital letter
- the terminals are in range of input alphabet and often represented by lowercase letters
- the sequence of substitutions to obtain a string is called a **derivation**

- any language that can be generated by some context-free grammar is called a **context-free language (CFL)**

3.2 Formal Definition Of a Context-Free Grammar

a **context-free grammar** is a 4-tuple (V, Σ, R, S) , where :

- V is a finite set called the **variables**
- Σ is a finite set, disjoint from V, called the **terminals**
- R is a finite set of **rules**,
- $S \in V$ is the start variable

- the **language of the grammar** is $\{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$

3.3 Ambiguity

if a grammar generates the same string in several different ways, we say that the string is derived **ambiguously** in that grammar . if a grammar generates some string **ambiguously**, we say that the grammar is **ambiguous**
when we say that a grammar generates a string ambiguously, we mean that the string has two different parse trees, not two different derivations .

- a string w is derived **ambiguously** in context-free grammar G if it has two or more different leftmost derivations . Grammar G is **ambiguous** if it generates some string ambiguously .

3.4 Chomsky Normal Form

in context-free grammars, it is convenient to have them in simplified form . One of the simplest and most useful forms is called the **Chomsky Normal Form** .

- a context-free grammar is in **Chomsky Normal Form** if every rule is of the form :

$$A \to BC$$
$$A \to a$$

where a is any terminal and A, B, and C are any variables — except that B and C may not be the start variable . in addition, we permit the rules $S \to \varepsilon$, where S is the start variable .

3.5 Theorem

- any context-free language is generated by a context-free grammar in Chomsky Normal Form .

4 Push Down Automata

PDA use different alphabet for its input and its stack , we specify input alphabet Σ and a stack alphabet Γ .
Note that : $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ and $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$

a **pushdown automaton** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q , Σ , Γ , and F are all finite sets , and

- Q is the set of states
- Σ is the input alphabet
- Γ is the stack alphabet
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q \times \Gamma_\varepsilon)$ is the transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accept states

4.1 Theorm

- a language is context-free if and only if some pushdown automaton recognizes it .

4.2 Lemma

- if a language is context-free, then some push down automaton recognizes it .

4.3 Lemma

- if a pushdown automaton recognized some language, then it is context-free

4.4 LR(k) grammar

Algorithms for $LR(k)$ grammars introduce *lookahead*.
The acronym LR(k) stands for :

- Left to right input processing
- Rightmost derivations(or equivalently, leftmost reductions)
- k symbols of *lookahead*

> - an **LR(k) grammar** is a context-free grammar such that the handle of every valid string is forced by lookahead k .

5 Turing Machine

The Turing machine model uses an infinite tape as its unlimited memory. It has a tape head that can read and write symbols and move around on the tape.
Initially the tape contains only the input string and is blank everywhere else. If the machine needs to store information, it may write this information on the tape. To read the information that it has written, the machine can move its head back over it. The machine continues computing until it decides to produce an output. The outputs accept and reject are obtained by entering designated accepting and rejecting states. If it doesn't enter an accepting or a rejecting state, it will go on forever, never halting.

The following list summarizes the differences between finite automata and Turing Machines .

1. a Turing Machine can both write on the tape and read from it
2. the read — write head can move both to the left and to the right
3. the tape is infinite
4. the special states for rejecting and accepting take effect immediately

5.1 Formal Definition of a Turing Machine

the heart of the definition of a Turing Machine is the transition function δ because it tells us how the machine gets from one step to the next .

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

when the machine is in certain state q and the head is over a tape square containing a symbol a, and if $\delta(q, a) = (r, b, L)$, the machine writes the symbol b replacing the a, and goes to the state r . L or R indicates whether the head moves to the Left or Right after writing . In this case, the L indicates a move to the Left .

a **Turing Machine** is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets

1. Q is teh set of states
2. Σ is the input alphabet not containing the **blank symbol**
3. Γ is the tape alphabet, where $b \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function
5. $q_0 \in Q$ is the start state
6. $q_{accept} \in Q$ is the accept state
7. $q_{reject} \in Q$ is the reject state , where $q_{reject} \neq q_{accept}$

- as a Turing Machine computes, changes occur in the current state, the current tape contents, and the current head location . a setting of these three items is called **configuration** of the Turing machine .