

# Formal Language and Automata Theory

Hsin-Jui, Chen

August, 2023

## Contents

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Regular Languages</b>                              | <b>2</b> |
| <b>2</b> | <b>Context-Free Languages</b>                         | <b>8</b> |
| 2.1      | Chomsky Normal Form . . . . .                         | 9        |
| 2.2      | Pushdown Automata and Context-Free Language . . . . . | 9        |

# 1 Regular Languages

**Definition 1** (Language). Let  $\Sigma$  be a finite set. Then a *string* over  $\Sigma$  is a finite sequence only consists of elements in  $\Sigma$ . A *formal language*  $L$  over the alphabet  $\Sigma$  is a subset of all strings (over  $\Sigma$ ).

**Definition 2** (Length). Let  $s$  be a string over  $\Sigma$ . We write  $|s|$  to denote the *length* of  $s$ , that is, the length of the finite sequence denoted by  $s$ .

In this section, we will define three kinds of automata.

**Definition 3** (Deterministic finite automaton). A *deterministic finite automaton*  $\mathcal{M}$  is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$ , which satisfies the following properties:

1.  $Q$  is the set of all *states*. We require that  $Q$  is non-empty and finite.
2.  $\Sigma$  denotes the *alphabet set* of all possible letters used in the input string. We require that  $\Sigma$  is a finite set.
3.  $\delta$  is the *transition function*, which is defined by

$$\begin{aligned}\delta : Q \times \Sigma &\rightarrow Q \\ (q, \sigma) &\mapsto q^+ = \delta(q, \sigma)\end{aligned}$$

Here  $q^+$  is the next state when the machine receive the input  $\sigma \in \Sigma$  at the state  $q \in Q$ .

4.  $q_0$  is an element in  $Q$ , which is called the *start state*.
5.  $F \subset Q$  is a subset of  $Q$ , which contains all *accepted states*.

In the following text, we shall write *DFA* to denote “deterministic finite automaton”.

**Definition 4** (Accepted string). Let  $\mathcal{M}$  be a given DFA. We say a string  $s = s_1 s_2 \cdots s_n$  (of length  $n$ ) over  $\Sigma$  is *accepted* (*recognized*) by  $\mathcal{M}$  if there is a sequence of states

$$\langle q_0, q_1, \dots, q_n \rangle$$

such that

$$q_i = \delta(q_{i-1}, s_i) \text{ for all } 1 \leq i \leq n,$$

and  $q_n \in F$ .

**Definition 5** (Language of a DFA). Let  $\mathcal{M}$  be a given DFA. Then  $L(\mathcal{M})$  denotes the set of all strings recognized by  $\mathcal{M}$ .

**Definition 6** (Regular language). Let  $L$  be a language over  $\Sigma$ . We say  $L$  is *regular*, if there is a DFA  $\mathcal{M}$  such that  $L = L(\mathcal{M})$ . In this case, we say  $L$  is recognized by the DFA  $\mathcal{M}$ .

We shall now define the concept of nondeterministic finite automaton. Although it seems quite powerful, it is actually equivalent to DFA.

**Definition 7** (Non-deterministic finite automaton). A *non-deterministic finite automaton*  $\mathcal{M}$  is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$ , which satisfies the following properties:

1.  $Q$  is the set of all states. We require that  $Q$  is non-empty and finite.
2.  $\Sigma$  denotes the alphabet set of all possible letters used in the input string. We require that  $\Sigma$  is a finite set.
3.  $\delta$  is the transition function, which is defined by

$$\begin{aligned}\delta : Q \times \Sigma_\epsilon &\rightarrow \mathcal{P}(Q) \\ (q, \sigma) &\mapsto Q^+ = \delta(q, \sigma)\end{aligned}$$

For clarity, we define  $\Sigma_\epsilon := \Sigma \cup \{\epsilon\}$  and  $\mathcal{P}(Q)$  is the power set of  $Q$ . Here  $Q^+$  is set of all possible next states when the machine receive the input  $\sigma \in \Sigma$  at the state  $q \in Q$ .

4.  $q_0$  is an element in  $Q$ , which is called the start state.
5.  $F \subset Q$  is a subset of  $Q$ , which contains all accepted states.

In the following text, we shall write *NFA* to denote “non-deterministic finite automaton”.

We also need to define what does it means when we say a string over  $\Sigma$  is accepted by the NFA  $\mathcal{M}$ .

**Definition 8** (Accepted string). Let  $\mathcal{M}$  be a given NFA. We say a string  $s = s_1 s_2 \cdots s_n$  ( $s_i \in \Sigma_\epsilon$ ) is *accepted* (*recognized*) by  $\mathcal{M}$  if there is a sequence of states

$$\langle q_0, q_1, \dots, q_n \rangle$$

such that

$$q_i \in \delta(q_{i-1}, s_i) \text{ for all } 1 \leq i \leq n,$$

and  $q_n \in F$ .

Then we could prove that DFA is equivalent to NFA in the following sense.

**Theorem 9.** *Let  $L$  be a language. Then  $L$  is recognized by a DFA if and only if  $L$  is recognized by an NFA.*

*Proof.* Suppose  $L$  is recognized by a DFA  $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ . Let  $\mathcal{M}' = (Q, \Sigma, \delta', q_0, F)$  be an NFA, whose transition function  $\delta'$  is defined by

$$\delta'(q, \sigma) = \begin{cases} \{\delta(q, \sigma)\} & \text{if } \sigma \in \Sigma \\ \emptyset & \text{if } \sigma = \epsilon \end{cases}.$$

It is clear that  $L(\mathcal{M}) = L(\mathcal{M}')$ .

Now suppose  $L$  is recognized by an NFA  $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ . Now for each state  $q \in Q$ , we define its  $E(q)$  be the set

$$E(q) := \{q\} \cup \{\text{the states can be reached from } q \text{ only by } \epsilon \text{ links.}\} = \bigcup_{k=0}^{\infty} \delta^{(k)}(q, \epsilon).$$

For each set  $A \in \mathcal{P}(Q)$ , we define

$$E(A) = \bigcup_{q \in A} E(q).$$

We now can give the definition of DFA that  $\mathcal{M}$  is equivalent to.

Let  $\mathcal{M}' = (\mathcal{P}(Q), \Sigma, \delta', E(q_0), F')$  be a DFA whose all possible states are  $\mathcal{P}(Q)$ , the start states is  $E(q_0)$ , and all the accepted states  $F' = \{X \in \mathcal{P}(Q) : X \cap F \neq \emptyset\}$ . It now remains to deal with the new transition function  $\delta'$ . We define

$$\delta'(A, \sigma) = \bigcup_{q \in A} E(\delta(q, \sigma))$$

for each  $A \in \mathcal{P}(Q)$  and  $\sigma \in \Sigma$ .

We have to show that  $L(\mathcal{M}) = L(\mathcal{M}')$ , however, this is not covered in class, therefore we only sketch the proof here. For a string  $s$ , we consider all states  $q_s$  in  $Q$  that can be reached from  $q_0$  by processing  $s$  (with respect to the machine  $\mathcal{M}$ ). We claim that  $\{q_s\}$  is the terminal state after  $\mathcal{M}'$  processes the string  $s$ . This can be proved by performing induction on the length  $|s|$  of  $s$ .  $\square$

**Definition 10** (Regular operation). Let  $L_1, L_2$ , and  $L$  are formal languages. *Regular operations* are the following three operations:

1.  $L_1 \cup L_2 = \{s : s \in L_1 \text{ or } s \in L_2\}$ .
2.  $L_1 \circ L_2 = \{st : s \in L_1 \text{ and } t \in L_2\}$ .
3.  $L^* = \{s : s = \sigma_1 \sigma_2 \cdots \sigma_n, n \geq 0 \text{ and } \sigma_i \in \Sigma\}$ . In other words,  $L^*$  is the language generated by  $L$ .

This is the definition covered in class, more formally, we may say a (binary) map

$$(L_1, L_2) \mapsto \varphi(L_1, L_2)$$

is a regular operation if  $\varphi(L_1, L_2)$  is a regular language whenever both  $L_1$  and  $L_2$  are regular. Similar definition can be extended to unary or even ternary operators (maps).

**Theorem 11.** *Regular operations preserves the regularity of languages, that is, if languages  $L_1, L_2$ , and  $L$  are regular, then  $L_1 \cup L_2, L_1 \circ L_2$ , and  $L^*$  are both regular.*

*Proof.* We might assume that both languages are using the same alphabet set, otherwise we may consider the union of those alphabet sets. Suppose  $\mathcal{M}_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ ,  $\mathcal{M}_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ , and  $\mathcal{M} = (Q, \Sigma, \delta, s, F)$  are NFAs that recognize  $L_1, L_2$ , and  $L$ , respectively. We may also assume that  $Q_1, Q_2$ , and  $Q$  are pairwise disjoint. Now consider a new NFA  $\mathcal{M}_{\text{un}} = (Q_{\text{un}}, \Sigma, \delta_{\text{un}}, s_{\text{un}}, F_{\text{un}})$ , where

1.  $Q_{\text{un}} := Q_1 \cup Q_2 \cup \{s_{\text{un}}\}$  is the set of states.
2.  $s_{\text{un}}$  is the start state.

3.  $F_{\text{un}} = F_1 \cup F_2$  is the set of accepted states.
4.  $\delta_{\text{un}}$  is the new transition function defined by

$$\delta_{\text{un}}(q, \sigma) = \begin{cases} \delta_i(q, \sigma) & \text{if } q \in Q_i \quad (i = 1 \text{ or } 2) \\ \{s_1, s_2\} & \text{if } q = s_{\text{un}} \text{ and } \sigma = \epsilon \\ \emptyset & \text{otherwise} \end{cases}.$$

It is clear that  $L(\mathcal{M}_{\text{un}}) = L_1 \cup L_2$ . We now let  $\mathcal{M}_{\text{cat}} = (Q_{\text{cat}}, \Sigma, \delta_{\text{cat}}, s_{\text{cat}}, F_{\text{cat}})$ , where

1.  $Q_{\text{cat}} = Q_1 \cup Q_2$  is the set of states.
2.  $s_{\text{cat}} = s_1$  is the start state.
3.  $F_{\text{cat}} = F_2$  is the set of accepted states.
4.  $\delta_{\text{cat}}$  is the new transition function defined by

$$\delta_{\text{cat}}(q, \sigma) = \begin{cases} \delta_1(q, \sigma) & \text{if } q \in Q_1 \setminus F_1 \\ \delta_2(q, \sigma) & \text{if } q \in Q_2 \\ \delta_1(q, \epsilon) \cup \{s_2\} & \text{if } q \in F_1 \text{ and } \sigma = \epsilon \\ \delta_1(q, \sigma) & \text{otherwise} \end{cases}.$$

It is clear that  $L(\mathcal{M}_{\text{cat}}) = L_1 \circ L_2$ . We now consider the NFA  $\mathcal{M}_* = (Q \cup \{s_*\}, \Sigma, \delta_*, s_*, F_*)$ , where

1.  $Q \cup \{s_*\}$  is the set of states.
2.  $s_*$  is the start state.
3.  $F_* := F \cup \{s_*\}$  is the set of accepted states.
4.  $\delta_*$  is the new transition function defined by

$$\delta_*(q, \sigma) = \begin{cases} \delta(q, \sigma) & \text{if } q \in Q \setminus F \\ \delta(q, \sigma) \cup \{s\} & \text{if } q \in F \text{ and } \sigma = \epsilon \\ \delta(q, \sigma) & \text{if } q \in F \text{ and } \sigma \neq \epsilon \\ \{s\} & \text{if } q = s_* \text{ and } \sigma = \epsilon \\ \emptyset & \text{if } q = s_* \text{ and } \sigma \neq \epsilon \end{cases}.$$

It is clear that  $L(\mathcal{M}_*) = L^*$ . Discussions above proves the theorem. □

Although the proof is not quite rigorous —there are still some details need to be handled, we omit the details here.

**Definition 12** (Regular expression). We say a language  $L$  is a *regular expression* if one of the following conditions are met

1.  $L = \emptyset$ , or  $L = \{\epsilon\}$ , or  $L = \{\sigma\}$  for some  $\sigma \in \Sigma$ .
2.  $L = L_1 \cup L_2$ , for some regular expressions  $L_1$  and  $L_2$ .
3.  $L = L_1 \circ L_2$ , for some regular expressions  $L_1$  and  $L_2$ .
4.  $L = L_0^*$ , for some regular expression  $L_0$ .

By this definition, it is reasonable to write  $\Sigma^*$  to denote the set of all strings over  $\Sigma$ .

**Theorem 13.** *A language  $L$  is regular if and only if it is a regular expression.*

Before proving this theorem, we might introduce another kind of finite automata, called generalized non-deterministic finite automata.

**Definition 14** (Generalized non-deterministic finite automaton). *A generalized non-deterministic finite automaton  $\mathcal{M}$  is a 5-tuple  $(Q, \Sigma, \delta, q_s, q_{ac})$ , which satisfies the following properties:*

1.  $Q$  is the set of all states. We require that  $Q$  is non-empty and finite. We require that  $q_s, q_{ac} \in Q$ .
2.  $\Sigma$  denotes the alphabet set of all possible letters used in the input string. We require that  $\Sigma$  is a finite set.
3.  $\delta$  is the transition function, which is defined by

$$\begin{aligned} \delta : Q \setminus \{q_{ac}\} \times Q \setminus \{q_s\} &\rightarrow \mathcal{R} \\ (q_i, q_j) &\mapsto L_{ij} = \delta(q_i, q_j) \end{aligned}$$

For clarity, we define  $\mathcal{R}$  is the set of all regular expressions over  $\Sigma$ .

4.  $q_0$  is an element in  $Q$ , which is called the start state.
5.  $F \subset Q$  is a subset of  $Q$ , which contains all accepted states.

In the following text, we shall write *GNFA* to denote “generalized non-deterministic finite automaton”.

We also can define the concept of accepted strings.

**Definition 15** (Accepted string). Let  $\mathcal{M} = (Q, \Sigma, \delta, q_s, q_{ac})$  be a given GNFA. We say a string  $s = s_1 s_2 \cdots s_n$  ( $s_i \in \Sigma^*$ ) is *accepted (recognized)* by  $\mathcal{M}$  if there is a sequence of states

$$\langle q_s = q_0, q_1, \dots, q_{n-1}, q_n = q_{ac} \rangle$$

such that

$$s_i \in \delta(q_{i-1}, q_i) \text{ for all } 1 \leq i \leq n.$$

It is worth noting that  $\delta(q_{i-1}, q_i)$  is a language by the definition.

**Lemma 1.** Let  $\mathcal{M} = (Q, \Sigma, \delta, q_s, q_{ac})$  be a GNFA with  $|Q| \geq 3$ . Define a new GNFA  $\mathcal{M}' = (Q', \Sigma, \delta', q_s, q_{ac})$  by removing a state  $q_{rip} \in Q \setminus \{q_s, q_{ac}\}$ . More precisely, the new machine  $\mathcal{M}'$  after the removal has to meet the conditions:

1.  $Q' = Q \setminus \{q_{rip}\}$ .
2. For any two states  $s, t \in Q'$ , we define

$$\delta'(s, t) = \delta(s, t) \cup \delta(s, q_{rip}) \circ \delta(q_{rip}, q_{rip})^* \circ \delta(q_{rip}, t).$$

Then  $L(\mathcal{M}) = L(\mathcal{M}')$ .

**Lemma 2.** Let  $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$  be a DFA. We now define a GNFA  $\mathcal{M}' = (Q \cup \{q_s, q_{ac}\}, \Sigma, \delta', q_s, q_{ac})$  by

$$\delta'(s, t) = \begin{cases} \{\sigma \in \Sigma : \delta(s, \sigma) = t\} & \text{if } s \in Q \text{ and } t \in Q \\ \{\epsilon\} & \text{if } s = q_s \text{ and } t = q_0 \\ \{\epsilon\} & \text{if } s \in F \text{ and } t = q_{ac} \\ \emptyset & \text{otherwise} \end{cases}$$

Then  $L(\mathcal{M}) = L(\mathcal{M}')$ .

Both two lemmas are easy to verify. We now can prove Theorem 13 with these two lemmas.

*Proof of Theorem 13.* By Definition 12 and Theorem 11, we easily see that a language is regular if it is regular expression. Now suppose  $L$  is a regular language, then by Lemma 2, there is a GNFA  $\mathcal{M} = (Q, \Sigma, \delta, q_s, q_{ac})$  that recognize  $L$ . By applying  $(|Q| - 2)$  times Lemma 1, we remove all the states in  $Q \setminus \{q_s, q_{ac}\}$ . Hence we get a GNFA  $\mathcal{M}_f = (\{q_s, q_{ac}\}, \Sigma, \delta_f, q_s, q_{ac})$  which only have two states  $\{q_s, q_{ac}\}$ . Thus,  $L(\mathcal{M}) = L(\mathcal{M}_f) = \delta_f(q_s, q_{ac}) \in \mathcal{R}$ .  $\square$

**Theorem 16** (Pumping Lemma). *Suppose  $L$  is a regular language over  $\Sigma$ . Then there exists  $p \in \mathbb{N}$  such that for all  $s \in L$  with  $|s| \geq p$ , there are strings  $x, y, z \in \Sigma^*$  such that  $s = xyz$ ,  $|y| > 0$ ,  $|xy| \leq p$ , and*

$$s = xy^kz \in L \quad \text{whenever} \quad k \in \mathbb{N} \cup \{0\}.$$

*In this case, we call  $p$  the pumping length.*

Although this theorem is called the pumping lemma, I still label it as a theorem according to its importance on solving problems.

*Proof.* Let  $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$  be a DFA that recognize  $L$ . We let  $p = |Q|$ . Now given any string  $s \in L$  with  $|s| \geq p$ . Write  $s = s_1s_2 \cdots s_n$  ( $s_i \in \Sigma$ ). By the definition, there is a sequence of states

$$\langle q_0, q_1, \dots, q_n \rangle$$

such that

$$q_i = \delta(q_{i-1}, s_i) \text{ for all } 1 \leq i \leq n,$$

and  $q_n \in F$ . Since  $n \geq p$ , it follows by the pigeon's hole principle that there have to be two indices  $0 \leq l < r \leq p$  such that  $q_l = q_r$ . It is now clear that the sequence

$$q_0 \xrightarrow{x} q_l \xrightarrow{y^k} q_r \xrightarrow{z} q_n$$

recognizes  $xy^kz$ , therefore  $xy^kz \in L$  for all  $k \in \mathbb{N} \cup \{0\}$ . Note that  $p \geq r > l$ , we obtain  $|xy| \leq p$  and  $|y| > 0$ .  $\square$

**Corollary.** Given a language  $L$  over  $\Sigma$ . If for all  $p \in \mathbb{N}$ , there exists a string  $s \in A$  with  $|s| \geq p$  such that for all  $x, y, z \in \Sigma^*$  satisfying  $s = xyz$ ,  $|y| > 0$ , and  $|xy| \leq p$ , there is a  $k \in \mathbb{N}$  such that  $xy^kz \notin L$ . Then we may conclude that  $L$  is not regular.

## 2 Context-Free Languages

**Definition 17** (Context-free grammars). A 4-tuple  $\mathcal{G} = (V, \Sigma, R, S)$  is said to be a *context-free grammar* if the following conditions are met:

1.  $V$  is the variable set. We require that  $V$  is finite and non-empty.
2.  $\Sigma$  is the terminal alphabet set. We also require that  $\Sigma$  is finite, non-empty and  $V \cap \Sigma = \emptyset$ .
3.  $R$  is the rules of the grammar.  $R$  is a finite subset of  $V \times (V \cup \Sigma)^*$ . If  $(v, s) \in R$ , it means we can replace  $v \in V$  with the string  $s$  when we make up a sentence. We often write  $v \rightarrow s$  for  $(v, s) \in R$ .
4.  $S \in V$  is the start variable.

In the following text, we shall write CFG to denote “context-free grammar”.

**Definition 18** (Context-free languages). Given a context-free grammar  $\mathcal{G} = (V, \Sigma, R, S)$ . Consider a string of the form  $uAv$ , where  $u, v \in (V \cup \Sigma)^*$  and  $A \in V$ . We write

$$uAv \Rightarrow u\bar{w}v \quad (w \in (V \cup \Sigma)^*)$$

if  $(A, w) \in R$ . In this case, we say  $uAv$  *directly yields*  $u\bar{w}v$  (with respect to the CFG  $\mathcal{G}$ ).

Now suppose we have two string  $u, v$  over  $(V \cup \Sigma)$ . We write  $u \xRightarrow{*} v$  if  $u = v$  or there are some  $u_i \in (V \cup \Sigma)^*$  ( $1 \leq i \leq n$ ,  $n \in \mathbb{N} \cup \{0\}$ ) such that

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_n \Rightarrow v.$$

We now define the *context-free language*  $L(\mathcal{G})$  of the CFG  $\mathcal{G}$  is the language

$$\{w \in \Sigma^* : S \xRightarrow{*} w\}.$$

We shall write CFL to denote “context-free language”.

We now give an example of a grammar.

**Example 19.** We consider the grammar

$$\langle \text{expr} \rangle \rightarrow \langle \text{expr} \rangle + \langle \text{expr} \rangle \mid \langle \text{expr} \rangle \times \langle \text{expr} \rangle \mid (\langle \text{expr} \rangle) \mid a,$$

where  $\langle \text{expr} \rangle$  is the only variable (therefore it is the start variable) and  $a$  is any terminal character. It



is clear that  $a \times a + a$  has two ways to generate, as follows:

$$\begin{aligned} \langle \text{expr} \rangle &\Rightarrow \langle \text{expr} \rangle + \langle \text{expr} \rangle \Rightarrow \langle \text{expr} \rangle \times \langle \text{expr} \rangle + \langle \text{expr} \rangle \\ &\Rightarrow a \times \langle \text{expr} \rangle + \langle \text{expr} \rangle \Rightarrow a \times a + \langle \text{expr} \rangle \Rightarrow a \times a + a \\ \langle \text{expr} \rangle &\Rightarrow \langle \text{expr} \rangle \times \langle \text{expr} \rangle \Rightarrow a \times \langle \text{expr} \rangle \\ &\Rightarrow a \times \langle \text{expr} \rangle + \langle \text{expr} \rangle \Rightarrow a \times a + \langle \text{expr} \rangle \Rightarrow a \times a + a \end{aligned}$$

People would like to interpret the first expression as  $(a \times a) + a$  and the second as  $a \times (a + a)$  according to the approach we generate the sentence (string).

This example shows the concept of ambiguity, in general, if a sentence can be generated by two different ways, then we may say the sentence is ambiguity. More rigorously, we have the following

**Definition 20** (Ambiguity). Let  $\mathcal{G}$  be a grammar. We say  $S = u_0 \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \dots \Rightarrow u_k$  is a *leftmost derivation* if for each  $u_i \Rightarrow u_{i+1}$ , we only replace the leftmost variable with a string over  $V \cup \Sigma$  by the rules  $R$ . A string  $s \in L(\mathcal{G})$  is said to be *ambiguous* if there are two different leftmost derivation  $S \xRightarrow{*} s$ . A grammar  $\mathcal{G}$  is called *ambiguous* if there is a ambiguous string  $s \in L(\mathcal{G})$ .

**Definition 21** (Inherently ambiguous). We say a context-free language  $L$  is *inherently ambiguous* if for any CFG  $\mathcal{G}$  that generates  $L$ ,  $\mathcal{G}$  is ambiguous.

## 2.1 Chomsky Normal Form

In this subsection, we discuss a simpler grammar called Chomsky normal form. Later we will see that each CFG is equivalent to a Chomsky normal form.

**Definition 22** (Chomsky normal form). A CFG  $\mathcal{G}$  is said to be in *Chomsky normal form* if every rule is of the form

$$A \rightarrow BC \quad \text{or} \quad A \rightarrow \sigma,$$

where  $\sigma \in \Sigma$  is a terminal character and  $A, B$ , and  $C$  are any variables —except that  $B$  and  $C$  may not be the start variable. In addition, we permit the rule  $S \rightarrow \epsilon$ .

**Theorem 23.** Any context-free language  $L$  is generated by CFG in Chomsky normal form.

*Proof.*

□

## 2.2 Pushdown Automata and Context-Free Language

In this subsection, we will introduce another automata which recognize CFLs. To put it simply, a pushdown automaton is an NFA with a stack memory. Strictly speaking, we have the following

**Definition 24** (Pushdown automaton). A 6-tuple  $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, F)$  is said to be a *pushdown automaton* if the following conditions are met:

1.  $Q$  is the set of all *states*. We require that  $Q$  is non-empty and finite.
2.  $\Sigma$  denotes the *alphabet set* of all possible letters used in the input string. We require that  $\Sigma$  is a finite set.
3.  $\Gamma$  denotes the *stack alphabet set* of all possible letters used in the stack memory. We require that  $\Gamma$  is a finite set.
4.  $\delta$  is the *transition function*, which has the domain and co-domain as

$$\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow \mathcal{P}(Q \times \Gamma_\epsilon)$$

where  $(q^+, \gamma') \in \delta(q, \sigma, \gamma)$  means that if the current state is  $q$ , the input is  $\sigma$  and the top of the stack is  $\gamma$ , then one of the possible operation is move to the state  $q^+$  and change the top of the stack from  $\gamma$  to  $\gamma'$ . If  $(q^+, \gamma') \in \delta(q, \sigma, \gamma)$ , we often write the transition  $q \rightarrow q^+$  with  $\sigma, \gamma \rightarrow \gamma'$  to denote the link.

5.  $q_0$  is an element in  $Q$ , which is called the *start state*.
6.  $F \subset Q$  is a subset of  $Q$ , which contains all *accepted states*.

In the following text, we shall write *PDA* to denote “pushdown automaton”.

Similarly, we have

**Definition 25** (Accepted string). Let  $\mathcal{M}$  be a given PDA. We say a string  $s = s_1 s_2 \cdots s_n$  ( $s_i$  is possibly  $\epsilon$ ) over  $\Sigma$  is *accepted* (*recognized*) by  $\mathcal{M}$  if there are a sequence of states  $\langle q_0, q_1, \dots, q_n \rangle$  and two sequence of characters  $\langle a_1, \dots, a_n \rangle$  and  $\langle b_1, \dots, b_n \rangle$  such that

1.  $q_n \in F$ , namely, the final state is accepted.
2.  $(q_i, b_i) \in \delta(q_{i-1}, s_i, a_i)$  for all  $i = 1, \dots, n$ .
3. There exist strings  $w_0, w_1, \dots, w_n \in \Gamma^*$  such that

$$w_{i-1} = a_i t_i \quad \text{and} \quad w_i = b_i t_i$$

for all  $i = 1, \dots, n$ . Note that it is possible that  $a_i$  or  $b_i$  is  $\epsilon$ .

**Definition 26** (Language of a DFA). Let  $\mathcal{M}$  be a given PDA. Then  $L(\mathcal{M})$  denotes the set of all strings recognized by  $\mathcal{M}$ .

**Lemma 3.** Let  $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, F)$  be a given PDA. Then there exists a PDA  $\mathcal{M}'$  recognizing the same language as  $\mathcal{M}$  such that

1.  $|F| = 1$ , that is, there is only a single accepted state.

2. The stack is empty before accepted. That is, we added another constraint

$$w_0 = w_n = \epsilon$$

on the requirements of the sequences  $(q_i)$ ,  $(a_i)$ , and  $(b_i)$  in Definition 25.

3. Each transition is a pop or push operation. That is, if  $(q^+, \gamma') \in \delta(q, \sigma, \gamma)$ , then we require that exactly one of the character  $\{\gamma, \gamma'\}$  has to be  $\epsilon$ .

**Theorem 27.** *A language  $L$  is a CFL if and only if  $L$  is recognize by a PDA  $\mathcal{M}$ .*