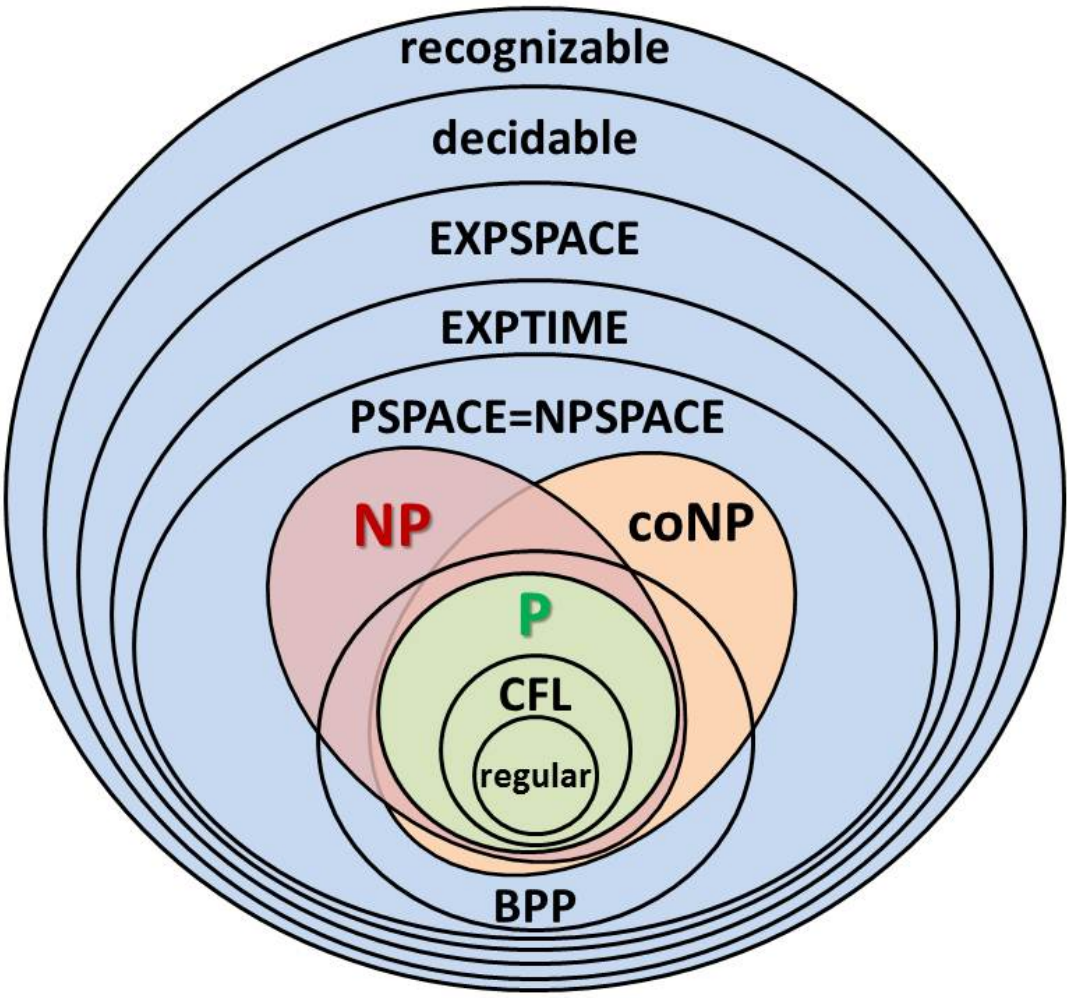
# Introduction to Computation Theory

Vinsong

November 11, 2025

**Abstract**

The lecture note of 2025 Fall Introduction to Computation Theory by professor 林智仁.



# Contents

Chapter 0

# Basic Knowledge

## Lecture 1

2025-09-01

## 0.1 Mathematical Notions

### 0.1.1 Set & its operation

**Definition 0.1.1** (Set). Omitted

**Definition** (Sequence & Tuple). Here are some definitions of basic containers

**Definition 0.1.2** (Sequence). Sequence is the objects in order, which have two properties:

- Order:

$$(1,2,3) \neq (2,1,3)$$

- Repetition:

$$\text{Sequence} : (1,2,3) \neq (1,1,2,3)$$

$$\text{Set} : \{1,2,3\} = \{1,1,2,3\}$$

**Definition 0.1.3** (Tuple). Finite sequence, $(1,2,3)$ is a 3-tuple

**Definition 0.1.4** (Cartesian Product). Here is the Cartesian Product between two sets. We define

$$A = \{1,2\},\ B = \{x,y\}$$

then,

$$A \times B = \{(1,x),(1,y),(2,x),(2,y)\}$$

### 0.1.2 Function & Relation

**Definition 0.1.5** (Function). Function is a machine with single output.

**Definition** (Equivalence Relations). Here are the properties of Equivalence Relations.

**Definition 0.1.6** (reflexive).

$$\forall x,\ xRx$$

**Definition 0.1.7** (symmetric).

$$\forall x, y,\ xRy \iff yRx$$

**Definition 0.1.8** (transitive).

$$xRy,\ yRz \implies xRz$$

**Example.**

$$i \equiv_7 j, \quad \text{if} \ \ 0 = i - j \mod 7$$

- Reflexive
$$i - i = 0 \mod 7$$
- Symmetric
$$i - j = 7a,\ j - i = -7a$$
- Transitive
$$i - j = 7a,\ j - k = 7b \implies i - k = 7(a + b)$$

### 0.1.3 String & Languages

**Definition** (String & Languages). Here is the definition of Language.

**Example** (Alphabet).

$$\{0, 1\}$$

**Example** (String).

$$01000$$

**Definition 0.1.9** (Language). Set of Strings

$$L(A)$$

is the language of $A$

## 0.2 Definitions, Theorems, and Proofs

- **Definition**: Introduce new concept.
- **Statement**: A sentence that is either true or flase.
- **Theorem**: A statement that is true.
  - **Lemma**: A "helping" theorem.
  - **Corllary**: A theorem that follows easily from another theorem.

### 0.2.1 Proof by Construction

**Proposition 0.2.1.** Sum of degrees of every graph is even

**Proof.** Each edge contributes 2 nodes, so

$$\sum_{v\in V} \deg(v) = 2 \times |E|$$

Hence, the sum of degrees of every graph is even. ■

**Note.** The implication is the definition of graphs.

### 0.2.2 Proof by Contradiction

Assume the statement is false, then deduce a contradiction.

### 0.2.3 Proof by Induction

- Basis: Prove for $n = 0$ or $n = 1$ or some trivial case.
- Inductive Step: Assume true for $n = k$ (Induction Hypothesis), prove for $n = k + 1$.

# Chapter 1

# Regular Languages

## 1.1 Deterministic Finite Automata (DFA)

- Automaton: single
- Automata: plural

**Definition 1.1.1** (Deterministic Finite Automata (DFA))**.** We define a DFA as a 5-tuple

$$(Q,\ \Sigma,\ \delta,\ q_0,\ F)$$

where

- $Q$: Set of states (Finite)
- $\Sigma$: Alphabet (i.e. set of input characters) (Finite)
- $\delta$: $Q \times \Sigma \to Q$: Transition Function
- $q_0 \in Q$: Start state
- $F \subset Q$: Set of accept states

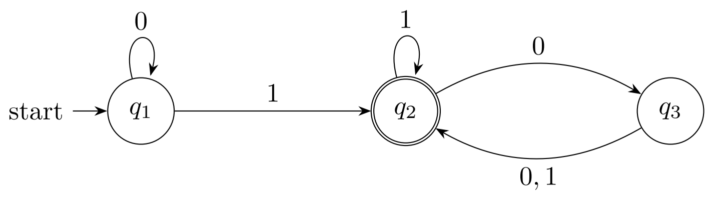


Figure 1.1: A state diagram

If we call this machine $M$, then we have.

$$M = (Q, \Sigma, \delta, q_0, F)$$

For the example given above,

$$Q = \{q_1, q_2, q_3\}$$
$$\Sigma = \{0, 1\}$$
$$q_0 = q_1$$
$$F = \{q_2\}$$

The $\delta$ function:

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

**Definition 1.1.2.** The language that recognize by a Machine $M$ is denoted as

$$L(M) = A$$

We say $A$ is recognizeed (accepted) by $M$.

## 1.1.1 Definition of Computation

Let,

- $M = (Q,\ \Sigma,\ \delta,\ q_0,\ F)$ be a finite automaton.
- $w = w_1, \cdots, w_n$ be a string over $\Sigma$.

**Theorem 1.1.1.** $M$ accepts $w$ if $\exists$ states $r_0 \cdots r_n$ such that

(1) $r_0 = q_0$

(2) $r_{i+1} = \delta(r_i, w_{i+1}), \quad i = [0, n-1]$

(3) $r_n \in F$

**Definition 1.1.3 (Regular Language).** A language is regular if recognized by some automata.

## 1.1.2 Regular Operations

**Definition.** Assume $A, B$ are given languages,

**Definition 1.1.4 (Union).**
$$A \cup B = \{w \mid w \in A \vee w \in B\}$$

**Definition 1.1.5 (Concatenation).**

$$A \circ B = \{w_1 w_2 \mid w_1 \in A, w_2 \in B\}$$

**Definition 1.1.6** (Kleene Star)**.**

$$A^* = \{w_1 \cdots w_k \mid k \geq 0, w_i \in A\}$$

which can also be defined as

$$\bigcup_{i=1}^{\infty} A_i = \{\varepsilon\} \cup A \cup A^2 \cup A^3 \cup \cdots, \quad A^0 = \{\varepsilon\},\ A^n = \{wv \mid w \in A^{n-1}, v \in A\}$$

**Definition 1.1.7** (closed)**.** We say an operation $R$ is closed if the following property holds if

$$x \in A, y \in A,\ \text{then } xRy \in A$$

**Theorem 1.1.2.** Regular languages are closed under the union, concatenation, and Kleene star.

**Proof.** We define two machines as follows

$$M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$$
$$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$$

if we union them, we can define a new machine

$$M_1 \cup M_2 = \begin{cases} M = (Q, \Sigma, \delta, q_0, F) \\ Q = \{(r_1, r_2) \mid r_1 \in Q_1, r_2 \in Q_2\} \\ \delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a)) \\ q_0 = (q_1, q_2) \\ F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\} \end{cases}$$

Hence, regular languages are closed under union. ■

## Lecture 2

2025-09-08

## 1.2 Nondeterministic Finite Automata (NFA)

First, we see a NFA that accept strings with 1 in 3rd position from the end,

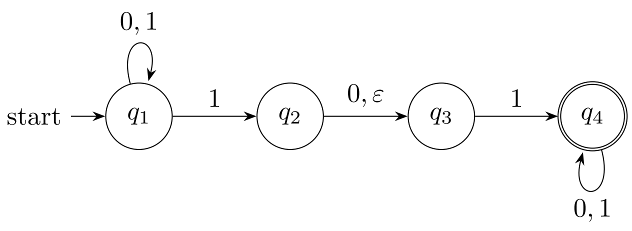


Figure 1.2: NFA machine

- $\delta$ is not a function, i.e. $\delta(q_1, 1) = q_1$ or $q_2$
- $\varepsilon$ between $q_2, q_3$ means $q_2$ can move to $q_3$ without any input

We can transport NFA to DFA by some method, for example, for the above NFA we can have:

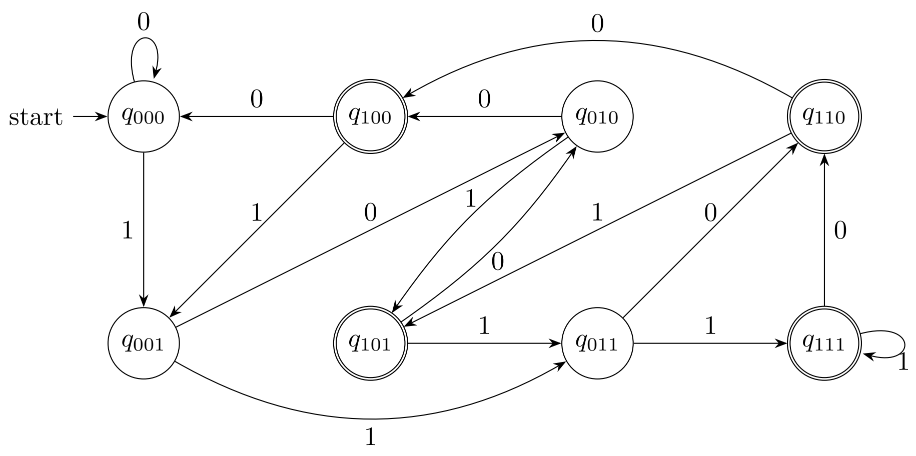


Figure 1.3: NFA machine transport to DFA

We can record it in three bits, it will be complicated.

**Definition 1.2.1** (power set).

$$P(Q) = \{X | X \in Q\}$$

which contain all the $2^{|Q|}$ combinations.

**Definition 1.2.2** (Nondeterministic Finite Automata (NFA)). We define a NFA as a 5-tuple

$$M = (Q, \Sigma_\varepsilon, \delta, q_0, F)$$

where

- $Q$: Set of states (Finite)
- $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$
- $\delta$: $Q \times \Sigma_\varepsilon \to P(Q)$
- $q_0 \in Q$
- $F \subset Q$

**Theorem 1.2.1.** We have $w$

$$w = y_1 \cdots y_m \quad \text{where } y_i \in \Sigma_\varepsilon$$

A sequence $r_0 \cdots r_m$ such that

(1) $r_0 = q_0$

(2) $r_{i+1} = \delta(r_i, y_{i+1}), \quad i = [0, n-1]$

(3) $r_n \in F$

**Note.** So $m$ may not be the original length (as $y_i$ may be $\varepsilon$)

### 1.2.1 Equivalence of DFA and NFA

From DFA $\Rightarrow$ NFA. Formally DFA is not an NFA due to $\Sigma$ and $\Sigma_\varepsilon$. but we can easily handle this by adding

$$q_i, \varepsilon \to \emptyset$$

For NFA $\Rightarrow$ DFA, we have the example on the slides on a graph.

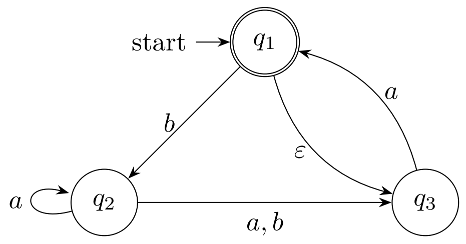


Figure 1.4: NFA example

$$\Downarrow$$

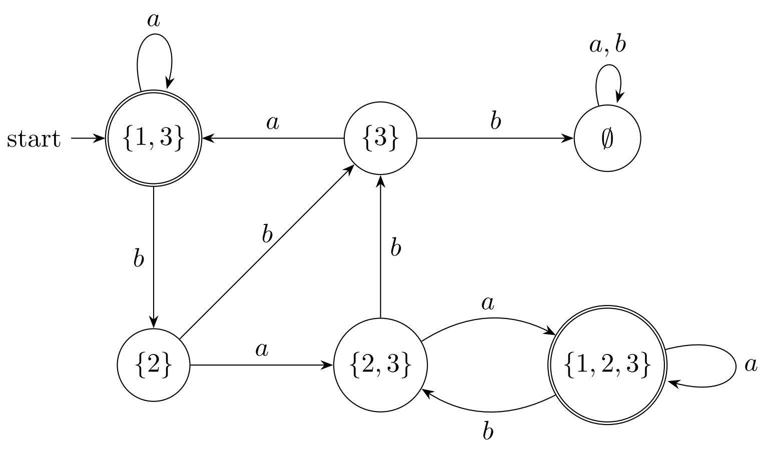


Figure 1.5: DFA convertion example

- Remove the states that are not reachable.
- Remove the states that not handle the $\varepsilon$ transition. For example, the start state

$$\{q_1\} \text{ wrong} \quad \to \quad \{q_1, q_3\} \text{ correct}$$

**Definition 1.2.3.**

$$E(\{q_0\}) = \{q_0\} \cup \{\text{states reached by } \varepsilon \text{ from } q_0\}$$

Then we can redefine the procedure formally.

**Theorem 1.2.2.** Given a NFA

$$M = (Q, \Sigma, \delta, q_0, F)$$

We can convert it to a DFA

$$M' = (Q', \Sigma, \delta', q_0', F')$$

where

- $Q' = P(Q)$
- $q_0' \in P(Q) = E(\{q_0\})$
- $F' = \{R \mid R \in Q', R \cap F \neq \emptyset\}$
- $\delta'$:

$$\delta'(R, a) = \bigcup_{r \in R} E(\delta(r, a))$$

### 1.2.2 Closure under reqular operations

We give two NFAs $N_1, N_2$,

$$N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$$
$$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$$

note that $\varepsilon \notin \Sigma$, and the graph of them are:

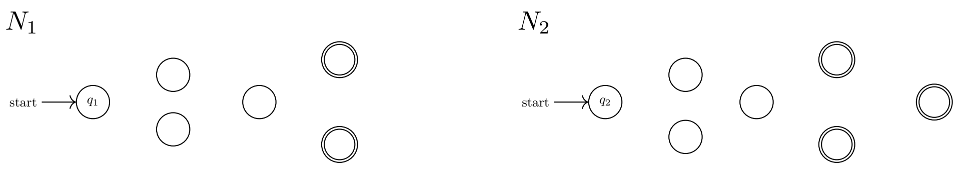


Figure 1.6: $N_1, N_2$

- **Union**: We can contrruct the $N_1 \cup N_2$ in

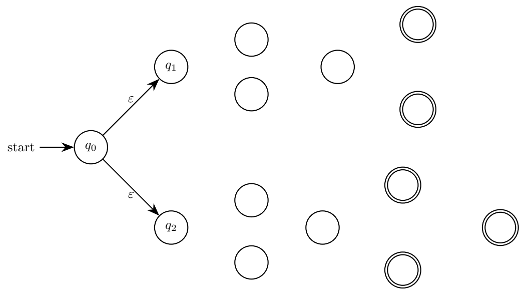


Figure 1.7: $N_1 \cup N_2$

**Proposition 1.2.1** (Construction of Union). New NFA is

$$N_1 \cup N_2 = (Q,\ \Sigma,\ \delta,\ q_0,\ F)$$

where

- $Q = Q_1 \cup Q_2 \cup \{q_0\}$
- $\delta$ :

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \\ \delta_2(q,a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0, a = \varepsilon \\ \emptyset & q = q_0, a \neq \varepsilon \end{cases}$$

- $F = F_1 \cup F_2$

- **Concatenation**: We can construct the $N_1 \circ N_2$ in

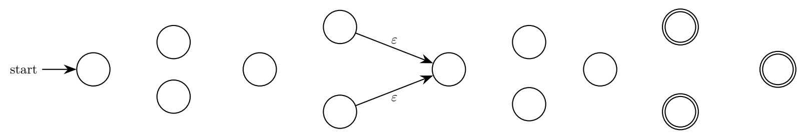


Figure 1.8: $N_1 \circ N_2$

**Proposition 1.2.2** (Construction of Concatenation)**.** New NFA is

$$N_1 \circ N_2 = (Q,\ \Sigma,\ \delta,\ q_0,\ F)$$

where

- $Q = Q_1 \cup Q_2$
- $\delta$ :

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1\ F_1 \\ \delta_2(q,a) & q \in Q_2 \\ \delta_1(q,\varepsilon) \cup \{q_2\} & q \in F_1, a = \varepsilon \\ \delta_1(q,\varepsilon) & q \in F_1, a \neq \varepsilon \end{cases}$$

- $q_0 = q_1$
- $F = F_2$

- **Kleene star**: $N_1^*$ can also accept $\{\emptyset\}$, then we can construct the $N_1^*$ in

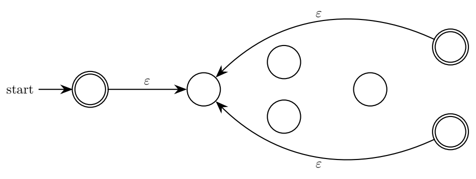


Figure 1.9: $N_1^*$

**Proposition 1.2.3** (Construction of Kleene Star)**.** New NFA is

$$N_1^* = (Q_1,\ \Sigma,\ \delta_1,\ q_0,\ F_1)$$

where

- $Q = Q_1 \cup \{q_0\}$

- $\delta$ :

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1\ F_1 \\ \delta_1(q,a) \cup \{q_1\} & q \in F_1, a = \varepsilon \\ \delta_1(q,\varepsilon) & q \in F_1, a \neq \varepsilon \\ \{q_1\} & q = q_0, a = \varepsilon \\ \emptyset & q = q_0, a \neq \varepsilon \end{cases}$$

- $F = F_1 \cup \{q_0\}$

**Note.** Some operations are also closed under regular languages,

- **Intersection:**

$$A_1 \cap A_2$$

Use the product automaton (the same construction as for Union). A string is accepted if and only if the state is in the accept states of both $N_1$ and $N_2$ at the same time.

- **Set Difference:**

$$A_1 - A_2$$

Use the product automaton as well. A string is accepted if the state is in the accept states of $N_1$ but *not* in the accept states of $N_2$.

- **Complement:**

$$A_1^c = \Sigma^* - A_1$$

Since $\Sigma^*$ is regular and the class of regular languages is closed under set difference, $A_1^c$ is also regular.

## Lecture 3

## 1.3 Regular expressions

2025-09-15

A regualar expression is a tool to describe a language.

**Definition 1.3.1 (Regular expressions).** $R$ is a regular expressions if it is one of the following expressions:

(1) $a$, where $a \in \Sigma$

(2) $\varepsilon$ ($\varepsilon \notin \Sigma$)

(3) $\emptyset$

(4) $R_1 \cup R_2$, where $R_1, R_2$ are regular expressions

(5) $R_1 \circ R_2$, where $R_1, R_2$ are regular expressions

(6) $R_1^*$, where $R_1$ is a regular expression

If their is no parentheses, we follow the order of:

$$\boxed{\texttt{Kleene star}} \rightarrow \boxed{\texttt{Concatenation}} \rightarrow \boxed{\texttt{Union}}$$

**Remark.**

$$R^+ = RR^*, \quad R^+ \cup \{\varepsilon\} = R^*$$

For $\emptyset$ and $\varepsilon$, we have

- $\varepsilon$: empty string
- $\emptyset$: empty language (language without any string)

$$\begin{aligned}
&(0 \cup \varepsilon)1^* = 01^* \cup 1^* \\
&(0 \cup \emptyset)1^* = 01^* \\
&\emptyset 1^* = 1^*\emptyset = \emptyset
\end{aligned}$$

**Example.** Here are some examples,

- Strings that start and end with the same symbol:
  $$0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$$
- $(\Sigma\Sigma)^*$: strings with even length
- $R \cup \emptyset = R$
- $R \circ \varepsilon = R$
- $\emptyset^* = \{\varepsilon\}$

Floating point numbers can also be represented by regular expressions. For example,

$$(+ \cup - \cup \varepsilon)(DD^* \cup DD^*.D^* \cup D^*.DD^*), \text{ where } D = \{0, \dots, 9\}$$

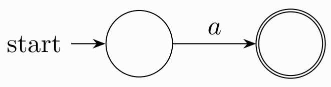
**Example.**

$$\begin{aligned}
&72 \in DD^* \\
&2.1 \in DD^*.D^* \\
&7. \in DD^*.D^* \\
&.01 \in D^*.DD^*
\end{aligned}$$

**Lemma 1.3.1.** Language by a regular expression $\Longrightarrow$ Regular (described by an automaton)

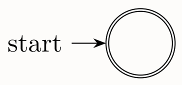
**Proof.** The proof is by induction,

- $R = a \in \Sigma$ can be recognize by



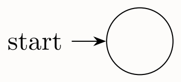
$$\begin{aligned}
&N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\}) \\
&\delta(q_1, a) = \{q_2\} \\
&\delta(r, b) = \emptyset, r \neq q_1 \text{ or } b \neq a
\end{aligned}$$

- $R = \varepsilon$



$$N = (\{q_1\}, \Sigma, \delta, q_1, \{q_1\})$$
$$\delta(q_1, a) = \emptyset, \forall a$$

- $R = \emptyset$



$$N = (\{q\}, \Sigma, \delta, q, \emptyset)$$
$$\delta(r, a) = \emptyset, \forall r, a$$

- $R = R_1 \cup R_2,\ R = R_1 \circ R_2,\ R = R_1^*$ have proof by NFA.

■

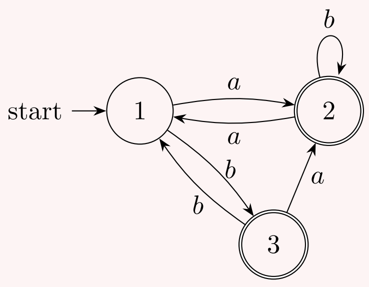
### 1.3.1 Convert a DFA to a regular expression

The idea is:

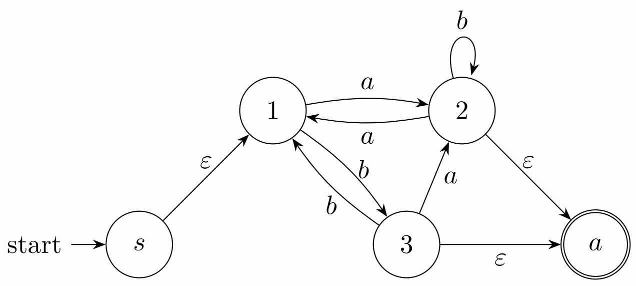
1° DFA $\longrightarrow$ GNFA

2° Remove states from GNFA until only the start and accept states.

**Question.** Convert the following DFA into regular expression.



**Answer.** First, convert to GNFA:



Next, is to remove the states one by one. We skip, so we can get the answer:

$$(a(aa \cup b)^*ab \cup b)((ba \cup a)(aa \cup b)^*ab \cup bb)^*((ba \cup a)(aa \cup b)^* \cup \varepsilon) \cup a(aa \cup b)^*$$

which is very complicated. ⊛

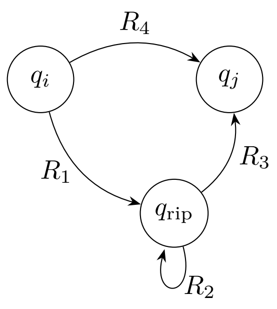
**Definition 1.3.2** (Generalized NFA(GNFA)). We define a GNFA as a 5-tuple

$$G = (Q, \Sigma, \delta, q_{start}, q_{accept})$$

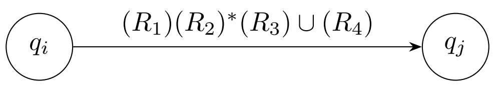
where

- $F$ is not a se, but a single accept state $q_{accept}$
- $\delta$ function is:
$$(Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \to R$$
  where $R$ is all regular expressions over $\Sigma$.
- Two new states:
$$q_{start} \to q_0 \text{ with } \varepsilon$$
$$\text{any } q \in F \to q_{accept} \text{ with } \varepsilon$$

Consider $q_{\text{rip}}$ is the state being removed



The new regular expression between $q_i$ and $q_j$ is



We can wrote the whole process into a algorithm.

**Algorithm 1.1:** CONVERT($G$) —State-Elimination from GNFA to RE

```
Input: $G = (Q, \Sigma, \delta, q_s, q_a)$ a GNFA
Output: A regular expression $R$ for the language of $G$
$k \leftarrow |Q|$;
;                                                    // number of states
if $k = 2$ then
    return $\delta(q_s, q_a)$ ;          // the (single) edge label from $q_s$ to $q_a$
Choose any $q_{\text{rip}} \in Q \setminus \{q_s, q_a\}$;
$Q' \leftarrow Q \setminus \{q_{\text{rip}}\}$;
Initialize $\delta'$ as the restriction of $\delta$ to $Q' \times Q'$;
foreach $q_i \in Q' \setminus \{q_a\}$ do
    foreach $q_j \in Q' \setminus \{q_s\}$ do
        $R_1 \leftarrow \delta(q_i, q_{\text{rip}})$;
        $R_2 \leftarrow \delta(q_{\text{rip}}, q_{\text{rip}})$;
        $R_3 \leftarrow \delta(q_{\text{rip}}, q_j)$;
        $R_4 \leftarrow \delta(q_i, q_j)$;
        $\delta'(q_i, q_j) \leftarrow R_4 \cup \big(R_1 R_2^* R_3\big)$;
$G' \leftarrow (Q', \Sigma, \delta', q_s, q_a)$;
return CONVERT($G'$);
```

# Lecture 4

2025-09-22

## 1.4 Pumping lemma

### 1.4.1 Non regular language

Some languages cannot be recognized by DFA such as,

$$\{0^n 1^n \mid n \geq 0\}$$

We might remember #0 first, but # of possible $n$'s is $\infty$, so we have some method to prove that the language is non-regular.

**Theorem 1.4.1** (pumping lemma). If $A$ is regular, $\exists p$ such that $\forall s \in A, |s| \geq p$,

$$\exists x, y, z, \text{ such that } s = xyz \text{ and}$$

1° $\forall i \geq 0, xy^i z \in A$

2° $|y| > 0$

3° $|xy| \leq p$

**Proof.** Skip, which is on the slides. ■

### 1.4.2 Example for Pumping Lemma

**Question.** Show that the language $L = \{0^n 1^n \mid n \geq 0\}$ is not regular using the pumping lemma.

**Answer.** Now consider the string

$$s = 0^p 1^p$$

We know that $|s| \geq p$. By the lemma, $s$ can be split into $xyz$ such that

$$xy^i z \in B, \forall i \geq 0, \quad |y| > 0, \quad \text{and } |xy| \leq p$$

1° If $y = 0 \cdots 0$, then

$$xy = 0 \cdots 0 \quad \text{and} \quad z = 0 \cdots 0\, 1 \cdots 1.$$

Thus,

$$xy^2 z : \#0 > \#1.$$

Hence $xy^2 z \notin B$, a contradiction.

2° If $y = 1 \cdots 1$, then similarly

$$xy^2 z \notin B \quad \text{as} \quad \#0 < \#1.$$

3° If $y = 0 \cdots 0\, 1 \cdots 1$, then

$$xy^2 z \notin B \quad \text{since it is not of the form } 0^* 1^*.$$

**Note.** Just pick one is sufficient to show the answer.

⊛

**Question.** Show that the language $C = \{w \mid \#0 = \#1\}$ is not regular using the pumping lemma.

**Answer.** We can use the situation in the pevious example, consider

$$s = 0^p 1^p$$

We can't proof the third condition due to $C = \{w \mid \#0 = \#1\}$ which just require the $\#0 = \#1$. Then we can use the third condition

$$|xy| \leq p$$

which means $y$ are strict into the first $0^p$ we can only consider the first case.

$$|xy| \leq p \Rightarrow y = 0 \cdots 0 \text{ in } s = 0^p 1^p$$

Then,

$$xy^2 z \notin C$$

⊛

**Lemma 1.4.1.** When using pumping lemma, we usually use contradiction, so we use

$$\forall p\ \exists s \in A,\ |s| \geq p,\ \Big[\forall x, y, z\ \Big((s = xyz \wedge |y| > 0 \wedge |xy| \leq p)\ \rightarrow\ \exists i \geq 0,\ xy^i z \notin A\Big)\Big].$$

Use the claim and the first, second condition to get the negation of the third condition.

**Question.** $D = \{1^{n^2} \mid n \geq 0\}$ is not regular

**Answer.** We pick

$$s = 1^{p^2} \in D$$

Then, if $s = xyz, |xy| \leq p, |y| > 0$, we can get

$$p^2 < |xy^2z| \leq p^2 + p \leq (p+1)^2$$

hence, $xy^2z \notin D$. ⊛

# Chapter 2

# Context-Free Languages

## Lecture 5

## 2.1 Context-Free Grammars (CFG)

2025-10-20

Which is more powerful, and can be used in compilers. A **Grammar** is a collection of substitution rules that describe the structure of a language.

**Example.** Consider a grammar $G_1$:

$$\begin{aligned} A &\to 0A1 \\ A &\to B \\ B &\to \# \end{aligned}$$

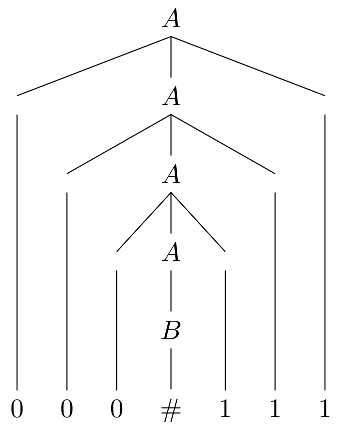
Here are the jargon terms:

- Each of one is called a **substitution rule**.
- **Variables** (non-terminals): $A, B$ (Capital letters)
- **Terminals**: $0, 1, \#$ (Lowercase letters, numbers, symbols)
- **Start variable**: $A$ (the variable we start with)

The process of generating strings is called **derivation**. $G_1$ generates 000#111 by

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$$

We can show the derivation using a **parse tree**:



### 2.1.1 Definition of CFG

The language of grammar $G$ is denoted by $L(G)$, for the language we discuss here,

$$L(G_1) = \{0^n \# 1^n \mid n \geq 0\}$$

Now we give the formal definition of CFG.

**Definition 2.1.1** (Context-Free Grammar). We defined a CFG as a 4-tuple

$$G = (V, \Sigma, R, S)$$

where

- $V$: Variables (Finite)
- $\Sigma$: Terminals (Finite)
- $R$: Rules:
$$\text{Variables} \to \text{Strings of Variables and Terminals (including } \varepsilon\text{)}$$
- $S \in V$: Start variable

For instance, for $G_1$,

$$G_1 = (\{A, B\}, \{0, 1, \#\}, R, A)$$

where $R$ is:

$$A \to 0A1 \mid B, \quad B \to \#$$

**Notation.** If $u, v, w$ are strings and rule $A \to w$ is applied, then we say

$$uAv \text{ yields } uwv$$

denoted as

$$uAv \Rightarrow uwv$$

**Notation.** If

$$u = v \text{ or } u \Rightarrow u_1 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v$$

then we write

$$v \overset{*}{\Rightarrow} u$$

**Definition 2.1.2** (Language of a CFG). The language generated by a CFG $G$ with start variable $S$ is

$$L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$$

### 2.1.2 Examples of CFGs

**Question.** Consider the grammar $G_2 = (\{S\}, \{a, b\}, R, S)$:

$$S \to aSb \mid SS \mid \varepsilon$$

What is $L(G_2)$?

**Answer.** If we let $a, b$ be the left and right parentheses respectively, then $L(G_2)$ is the set of all balanced parentheses. ⊛

**Example.** Consider the grammar $G_3 = (V, \Sigma, R, S)$ where

- $V = \{\langle \text{expr} \rangle, \langle \text{term} \rangle, \langle \text{factor} \rangle\}$
- $\Sigma = \{+, \times, (, ), a\}$
- $R$:

$$\begin{aligned} \langle \text{expr} \rangle &\to \langle \text{term} \rangle + \langle \text{expr} \rangle \mid \langle \text{term} \rangle \\ \langle \text{term} \rangle &\to \langle \text{factor} \rangle \times \langle \text{term} \rangle \mid \langle \text{factor} \rangle \\ \langle \text{factor} \rangle &\to (\langle \text{expr} \rangle) \mid a \end{aligned}$$

Consider the string $a + a \times a$:

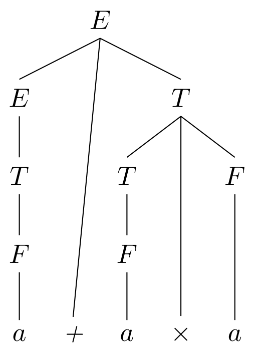


Figure 2.1: Parse tree of $a + a \times a$

Consider the string $(a + a) \times a$:

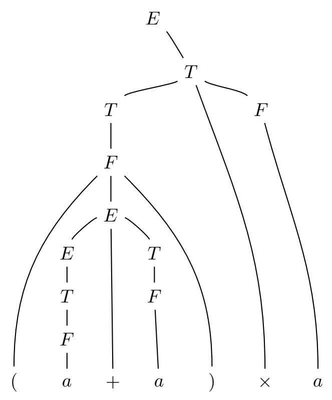


Figure 2.2: Parse tree of $(a + a) \times a$

**Note.** The example above shows that CFGs can express operator precedence and associativity.

### 2.1.3 Design of CFGs

We can design CFGs in many methods. Here are some common patterns:

- Combining smaller parts:

  **Example.** $L(G) = \{a^n b^n \mid n \geq 0\} \cup \{b^n a^n \mid n \geq 0\}$

  We can let the rule $R$ be:

  $$\begin{aligned} S_1 &\to aS_1b \mid \varepsilon \\ S_2 &\to bS_2a \mid \varepsilon \\ S &\to S_1 \mid S_2 \end{aligned}$$

- From DFA:

  **Lemma 2.1.1.** For any regular language $A$, there exists a CFG $G$ such that $L(G) = A$. The rules of CFG can be

  $$\begin{aligned} &R_i \to aR_j \quad \text{for each transition } \delta(q_i, a) = q_j \\ &R_i \to \varepsilon \quad \text{if } q_i \in F \end{aligned}$$

  The difference is that CFG allows the format

  $$R_i \to aR_jb$$

  But DFA only allows

  $$R_i \to aR_j$$

  where we treat $R_i$ as the state and let $\delta(R_i, a) = R_j$.

### 2.1.4 Parse Trees and Ambiguity

If we let the rules of $G_3$ be

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

We can see the following two parse trees for $a + a \times a$:

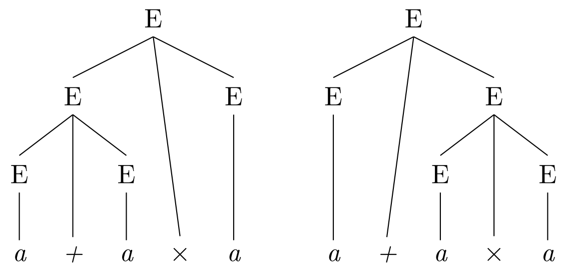


Figure 2.3: Two different parse trees for $a + a \times a$ under ambiguous grammar

This is called **ambiguity**. A CFG is **ambiguous** if there exists some string with two or more different parse trees. The above $G_3$ is **unambiguous**, $G_3'$ with new rules is **ambiguous**.

However, an unambiguous grammar may also generate same parse tree but different derivations. Consider $G_3$:

- We can do derivation

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

- We can also do derivation

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

which is not considered ambiguous. So we have the following definition:

**Definition 2.1.3** (leftmost derivation)**.** A **leftmost derivation** is a derivation where at each step, the leftmost variable is replaced.

Then we can have the formal definition of ambiguity:

**Definition 2.1.4** (Ambiguous)**.** $A$ is **ambiguous** if $w \in A$ and there exists two or more different leftmost derivations for $w$.

**Definition 2.1.5** (Inherent Ambiguity)**.** A language is **inherently ambiguous** if it only has ambiguous grammars.

**Example.** Consider the language

$$L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$$

We can consider the string $a^2b^2c^2$. It can be generated by two different leftmost derivations. First we consider

$$S \Rightarrow S_1 \mid S_2$$

- Using $i = j$:

$$\begin{aligned}S_1 &\to AC\\ A &\to aAb \mid \varepsilon\\ C &\to cC \mid \varepsilon\end{aligned}$$

  the derivation is

$$S_1 \Rightarrow AC \Rightarrow aAbC \Rightarrow aaAbbC \Rightarrow aabbC \Rightarrow aabbcC \Rightarrow aabbcc$$

- Using $j = k$:

$$\begin{aligned}S_2 &\to A'C'\\ A' &\to a\,A' \mid \varepsilon\\ C' &\to b\,C'\,c \mid \varepsilon\end{aligned}$$

  the derivation is

$$S_2 \Rightarrow A'C' \Rightarrow aA'C' \Rightarrow aaA'bC'c \Rightarrow aabbC'cc \Rightarrow aabbcc$$

## Lecture 6

2025-10-27

## 2.2 Chomsky Normal Form

We want to simplify the structure of context-free grammars. One useful normal form is the Chomsky Normal Form (CNF).

**Definition 2.2.1** (Chomsky Normal Form). A context-free grammar is in **Chomsky Normal Form** if all its production rules are of the form:

- $A \to BC$, where $A, B, C$ are non-terminal symbols and $B, C$ are not the start symbol.
- $A \to a$, where $a \in \Sigma$ ($\varepsilon \notin \Sigma$)
- $S \to \varepsilon$ is allowed, where $S$ is the start symbol.

**Example.** Convert the following CFG to CNF:

$$\begin{aligned} S &\to ASA \mid aB \\ A &\to B \mid S \\ B &\to b \mid \varepsilon \end{aligned}$$

First, we add $S_0$ as the new start symbol:

$$S_0 \to S \quad S \to ASA \mid aB \quad A \to B \mid S \quad B \to b \mid \varepsilon$$

Next, we remove the $\varepsilon$-productions $B \to \varepsilon$:

$$S_0 \to S \quad S \to ASA \mid aB \mid a \quad A \to B \mid \varepsilon \mid S \quad B \to b$$

Next, we remove the $\varepsilon$-productions $A \to \varepsilon$:

$$S_0 \to S \quad S \to ASA \mid aB \mid a \mid AS \mid SA \mid S \quad A \to B \mid S \quad B \to b$$

Next, we remove single production $S \to S$:

$$S_0 \to S \quad S \to ASA \mid aB \mid a \mid AS \mid SA \quad A \to B \mid S \quad B \to b$$

Next, we remove single production $S_0 \to S$:

$$S_0 \to ASA \mid aB \mid a \mid AS \mid SA \quad S \to ASA \mid aB \mid a \mid AS \mid SA \quad A \to B \mid S \quad B \to b$$

Next, we remove single production $A \to B$, $A \to S$:

$$S_0 \to ASA \mid aB \mid a \mid AS \mid SA \quad S \to ASA \mid aB \mid a \mid AS \mid SA \quad A \to b \mid ASA \mid aB \mid a \mid AS \mid SA \quad B \to b$$

Finally, we convert to CNF by introducing new variables for terminals and breaking down long productions:

$$\begin{aligned} S_0 &\to AA_1 \mid UB \mid a \mid AS \mid SA \\ S &\to AA_1 \mid UB \mid a \mid AS \mid SA \\ A &\to b \mid AA_1 \mid UB \mid a \mid AS \mid SA \\ A_1 &\to SA \\ B &\to b \\ U &\to a \end{aligned}$$

### 2.2.1 Procedure of Converting CFG to CNF

To convert any CFG to CNF, we can follow these steps:

1° **Add** a new start symbol $S_0$ with the production

$$S_0 \to S$$

2° **Remove** all $\varepsilon$-productions, except for the start symbol, i.e. $A \to \varepsilon$ $(A \neq S_0)$, for any

$$\cdots \to uAv$$

add the production

$$\cdots \to uv$$

3° **Remove** single productions of $A \to B$ where $A, B \in V/\{S\}$.

$$A \to B,\ B \to \gamma \quad \Rightarrow \quad A \to \gamma$$

> **Remark.** $A \to \gamma$ can't be a unit rule previously removed.

4° **Convert** remaining productions to CNF:

$$A \to u_1 u_2 \cdots u_k \quad u_i \in V \cup \Sigma$$

and

$$\text{if } k = 1, \text{ then } u_i \in \Sigma$$

Convert as follows:

$$\begin{aligned} A &\to u_1 A_1 \\ A_1 &\to u_2 A_2 \\ &\vdots \end{aligned}$$

Replaced every terminal $u_i \in \Sigma$ with a new variable $U_i$:

$$U_i \to u_i \quad u_i \in \Sigma$$

### 2.2.2 Infinite Loop in Converting

> **Example.** Consider the grammar:
>
> $$\begin{aligned} S &\to B \mid \varepsilon \\ B &\to S \mid \varepsilon \end{aligned}$$

We first add a new start symbol:

$$S_0 \to S \quad S \to B \mid \varepsilon \quad B \to S \mid \varepsilon$$

Next, we remove the $\varepsilon$-productions:

$$S_0 \to S \mid \varepsilon \quad S \to B \quad B \to S \mid \varepsilon$$

Next, we remove the $\varepsilon$-productions again:

$$S_0 \to S \mid \varepsilon \quad S \to B \mid \varepsilon \quad B \to S$$

This process will continue indefinitely. The reason is $S \to \varepsilon$ has been handled. So there is no need to add $S \to \varepsilon$.

## 2.3 Pushdown Automata

We now introduce the machine that recognizes context-free languages (CFL), called Pushdown Automata (PDA). PDA is a machine with a stack, which is a way to store previous states.

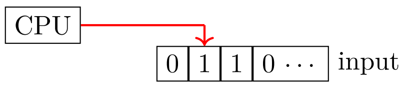


Figure 2.4: DFA or NFA

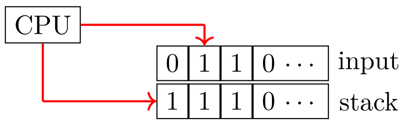


Figure 2.5: Pushdown Automata (PDA)

**Example.** Consider the language $A = \{0^n1^n \mid n \geq 0\}$. We can design a PDA to recognize $A$:

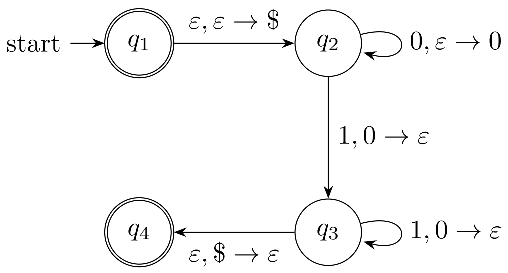


Figure 2.6: PDA for $A = \{0^n1^n \mid n \geq 0\}$

\$ is a special bottom stack symbol to indicate the initial state of the stack. The PDA works as follows:

- $q_2 \to q_2$, put 0 into stack
- $q_2 \to q_3$ and $q_3 \to q_3$, read 1 and `pop` 0 up

If the input is 0011 which is same as $\varepsilon 0011 \varepsilon$, the process is as follows:

$$\begin{aligned}
&q_1, \emptyset, \varepsilon \\
&q_2, \{\$\}, 0 \\
&q_2, \{0, \$\}, 0 \\
&q_2, \{0, 0, \$\}, 1 \\
&q_3, \{0, \$\}, 1 \\
&q_3, \{\$\}, \varepsilon \\
&q_4, \{\}
\end{aligned}$$

**Notation.** $\{\}$: contents of the stack before processing the input character.

### 2.3.1 Formal definition of PDA

**Definition 2.3.1 (Pushdown Automata).** A **pushdown automaton** (PDA) is a 6-tuple

$$(Q, \Sigma, \Gamma, \delta, q_0, F)$$

, where

- $Q$: States
- $\Sigma$: Input alphabet
- $\Gamma$: Stack alphabet
- $\delta$: Transition function
$$Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$$
- $q_0 \in Q$: Start state
- $F \subset Q$: Set of accepting states

The definition of the above PDA for $A = \{0^n1^n \mid n \geq 0\}$ is as follows:

- $Q = \{q_1, q_2, q_3, q_4\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, \$\}$
- $q_0 = q_1$
- $F = \{q_1, q_4\}$

For the the transition function, we care about three things:

- Current state
- Current input
- Top of the stack

The transition function $\delta$ works as follows:

| | 0 | | | 1 | | | $\varepsilon$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | \$ | $\varepsilon$ | 0 | \$ | $\varepsilon$ | 0 | \$ | $\varepsilon$ |
| $q_1$ | | | | | | | | | $\{(q_2, \$)\}$ |
| $q_2$ | | | $\{(q_2, 0)\}$ | $\{(q_3, \varepsilon)\}$ | | | | | |
| $q_3$ | | | | $\{(q_3, \varepsilon)\}$ | | | $\{(q_4, \varepsilon)\}$ | | |
| $q_4$ | | | | | | | | | |

For example, we say the transition of $q_2 \to q_3$ to be

$$\delta(q_2, 1, 0) = \{(q_3, \varepsilon)\}$$

### 2.3.2 Nondeterministic situation

**Example.** Design a PDA for the language $B = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } j = k\}$.

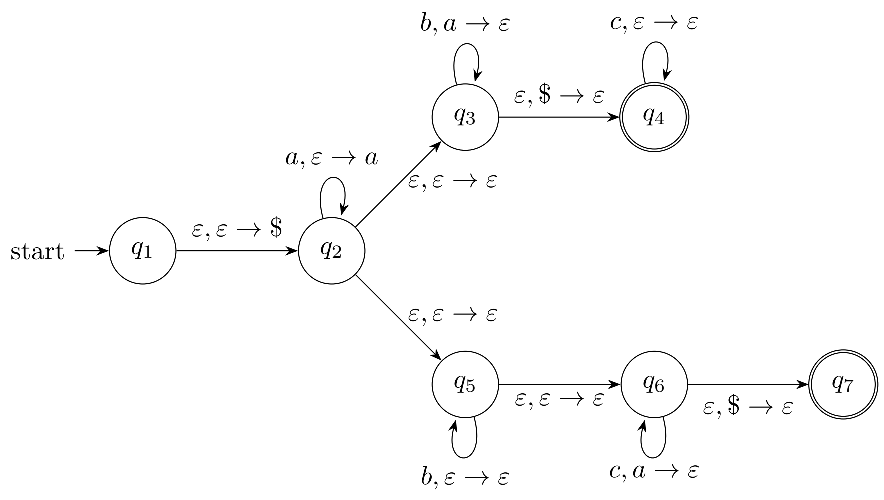
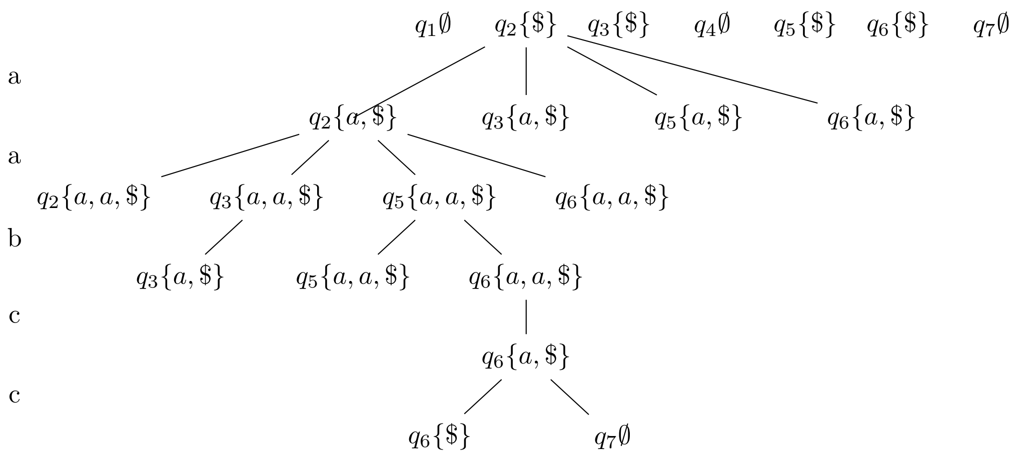

$b, a \to \varepsilon$
$c, \varepsilon \to \varepsilon$
$\varepsilon, \$ \to \varepsilon$
$q_3$
$q_4$
$a, \varepsilon \to a$
$\varepsilon, \varepsilon \to \varepsilon$
start
$q_1$
$\varepsilon, \varepsilon \to \$$
$q_2$
$\varepsilon, \varepsilon \to \varepsilon$
$q_5$
$q_6$
$q_7$
$\varepsilon, \varepsilon \to \varepsilon$
$\varepsilon, \$ \to \varepsilon$
$b, \varepsilon \to \varepsilon$
$c, a \to \varepsilon$

Figure 2.7: Nondeterministic PDA

We input $a^2bc^2$, to illustrate the process, we can build the following computation tree:


$q_1\emptyset$ $q_2\{\$\}$ $q_3\{\$\}$ $q_4\emptyset$ $q_5\{\$\}$ $q_6\{\$\}$ $q_7\emptyset$
a
$q_2\{a,\$\}$ $q_3\{a,\$\}$ $q_5\{a,\$\}$ $q_6\{a,\$\}$
a
$q_2\{a,a,\$\}$ $q_3\{a,a,\$\}$ $q_5\{a,a,\$\}$ $q_6\{a,a,\$\}$
b
$q_3\{a,\$\}$ $q_5\{a,a,\$\}$ $q_6\{a,a,\$\}$
c
$q_6\{a,\$\}$
c
$q_6\{\$\}$ $q_7\emptyset$

**Example.** Design a PDA for the language $C = \{ww^R \mid w \in \{0,1\}^*\}$.

**Idea.** Symbols pushed to stack, nondeterministically guess middle is reached

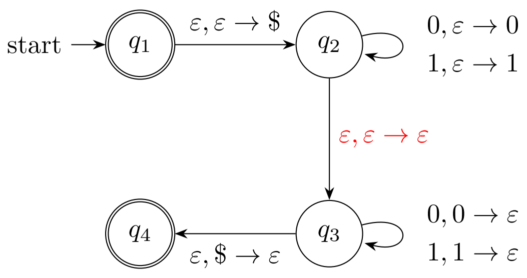


Figure 2.8: PDA for $C = \{ww^R \mid w \in \{0,1\}^*\}$

### 2.3.3 Converting CFL to PDA

**Example.** Convert the CFG $G$ to PDA that recognizes $L(G)$:

$$S \to aTb \mid b$$
$$T \to Ta \mid \varepsilon$$

**Idea.** For rule substitution, we replace the left-hand side variable with the right-hand side string i.e.

$$A \to \gamma \quad \Rightarrow \quad \texttt{pop}\ A \text{ from stack, } \texttt{push}\ \gamma \text{ to stack}$$

if there are multiple productions for $A$, we `push` them in a reversed way.

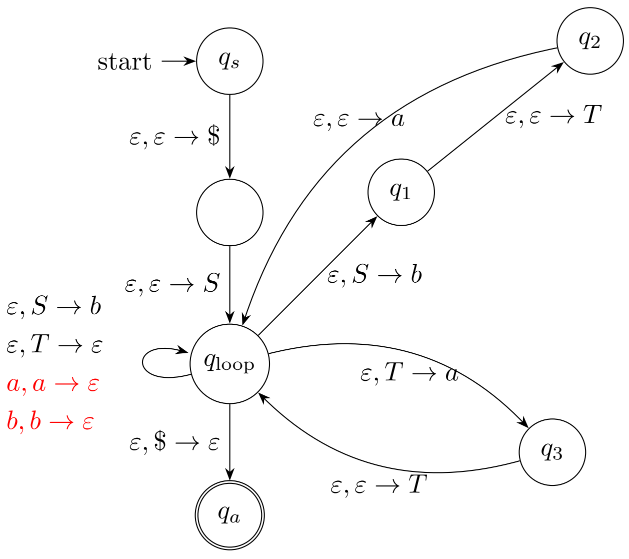

start
$q_s$
$\varepsilon, \varepsilon \to \$$
$\varepsilon, \varepsilon \to S$
$\varepsilon, S \to b$
$\varepsilon, T \to \varepsilon$
$a, a \to \varepsilon$
$b, b \to \varepsilon$
$q_{\text{loop}}$
$\varepsilon, \$ \to \varepsilon$
$q_a$
$\varepsilon, S \to b$
$q_1$
$\varepsilon, \varepsilon \to T$
$q_2$
$\varepsilon, \varepsilon \to a$
$\varepsilon, T \to a$
$q_3$
$\varepsilon, \varepsilon \to T$

Figure 2.9: PDA for CFG $G$

**Remark.** There are two transitions we must add to process the "input":

$$a, a \to \varepsilon$$
$$b, b \to \varepsilon$$

The procedure of converting CFG to PDA is as follows:

$$\begin{aligned}
&q_{\text{start}} \xrightarrow{\varepsilon} q_{\text{loop}}, \{S, \$\} \xrightarrow{\varepsilon} q_1, \{b, \$\} \xrightarrow{\varepsilon} q_2, \{T, b, \$\} \\
&\xrightarrow{\varepsilon} q_{\text{loop}}, \{a, T, b, \$\} \xrightarrow{a} q_{\text{loop}}, \{T, b, \$\} \\
&\xrightarrow{\varepsilon} q_3, \{a, b, \$\} \xrightarrow{\varepsilon} q_{\text{loop}}, \{T, a, b, \$\} \\
&\xrightarrow{\varepsilon} q_3, \{a, a, b, \$\} \xrightarrow{\varepsilon} q_{\text{loop}}, \{T, a, a, b, \$\} \\
&\xrightarrow{\varepsilon} q_3, \{a, a, a, b, \$\} \xrightarrow{\varepsilon} q_{\text{loop}}, \{T, a, a, a, b, \$\} \\
&\xrightarrow{\varepsilon} q_{\text{loop}}, \{a, a, a, b, \$\} \xrightarrow{a} q_{\text{loop}}, \{a, a, b, \$\} \\
&\xrightarrow{a} q_{\text{loop}}, \{a, b, \$\} \xrightarrow{a} q_{\text{loop}}, \{b, \$\} \\
&\xrightarrow{b} q_{\text{loop}}, \{\$\} \xrightarrow{\varepsilon} q_{accept}
\end{aligned}$$

**Proposition 2.3.1.** Even with a non-deterministic setting, we ensure that only strings generated by this CFG can be accepted by the PDA

- A string is accepted only if all characters are processed (this is part of the PDA definition!)
- We have \$ to ensure that the stack is empty in the end

### 2.3.4 Converting PDA to CFL

**Lemma 2.3.1.** Language recognized by PDA $\implies$ context free

**Note.** We need PDA to satisfy

1° Single start state

2° Stack empty before accepting

3° Each transition `push` or `pop`, but not both

**Idea.** For each pair of states $p, q \in Q$ of a PDA $P$, we have $A_{pq}$ and

$$A_{pq} \text{ generates } x \Rightarrow P \text{ from } p \text{ with empty stack to } q \text{ with empty stack, reading } x$$

First, we discuss how to handle transitions

$$\forall p, q, r \in Q,\ A_{pq} \to A_{pr}A_{rq}$$

We let the

- $x$-axis: input string
- $y$-axis: stack height

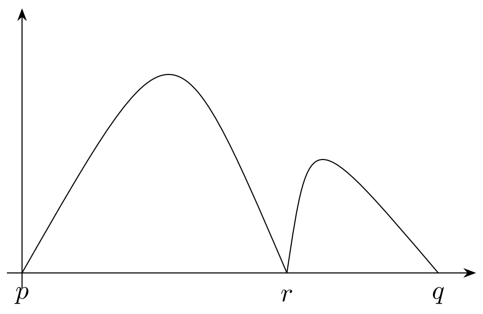


Figure 2.10: PDA transition $A_{pq} \to A_{pr}A_{rq}$

If we can go

$$\text{from } p \text{ to } r \text{ without changing stack}$$

and

$$\text{from } r \text{ to } q \text{ without changing stack}$$

then we can do

$$\text{from } p \text{ to } q \text{ without changing stack}$$

Next, we have

$$\forall p, q, r, s \in Q,\ a, b \in \Sigma_\varepsilon,\ t \in \Gamma$$

If,

$$(r, t) \in \delta(p, a, \varepsilon) \text{ and } (q, \varepsilon) \in \delta(s, b, t)$$

we discuss how to handle transitions

$$A_{pq} \to aA_{rs}b$$

Then we have

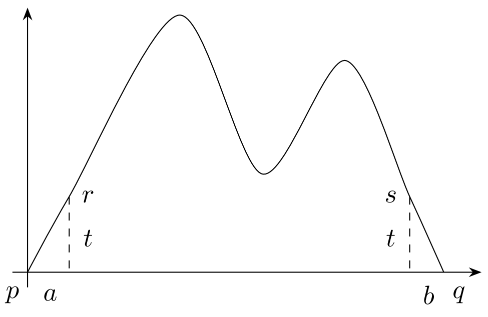


Figure 2.11: PDA transition $A_{pq} \to aA_{rs}b$

Finally, we have the following base case:

$$\forall p \in Q,\ A_{pp} \to \varepsilon$$

To follow the condition (1°), we give a new example

**Example.** Consider the language $L = \{0^n1^n \mid n \geq 1\}$.

Now $q_1$ is not an accept state

$$\begin{tikzcd} \text{start} \arrow[r] & q_1 \arrow[r, "\varepsilon,\varepsilon\to\$"] & q_2 \arrow[loop right, "0,\varepsilon\to 0"] \arrow[d, "1,0\to\varepsilon"] \\ & q_4 & q_3 \arrow[l, "\varepsilon,\$\to\varepsilon"] \arrow[loop right, "1,0\to\varepsilon"] \end{tikzcd}$$

Figure 2.12: PDA for $A = \{0^n1^n \mid n \geq 1\}$

Consider two elements in $\Gamma$

$$t_0 = \$, \quad t_1 = 0$$

- $t = \$$

| p | r | s | q | t | a | b |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | $\$$ | $\varepsilon$ | $\varepsilon$ |

then we can get the rule

$$A_{14} \to A_{23}$$

- $t = 0$

| p | r | s | q | t | a | b |
|---|---|---|---|---|---|---|
| 2 | 2 | 2 | 3 | 0 | 0 | 1 |
| 2 | 2 | 3 | 3 | 0 | 0 | 1 |

  then we can get the rules

$$A_{23} \to 0A_{22}1$$
$$A_{23} \to 0A_{23}1$$

Other rules: 64 rules

$$A_{11} \to A_{11}A_{11}$$
$$A_{11} \to A_{12}A_{21}$$
$$A_{11} \to A_{13}A_{31}$$
$$A_{11} \to A_{14}A_{41}$$
$$\vdots$$

and

$$A_{11} \to \varepsilon$$
$$A_{22} \to \varepsilon$$
$$A_{33} \to \varepsilon$$
$$A_{44} \to \varepsilon$$

### 2.3.5 Procedure of converting PDA to CFL

**Proposition 2.3.2.** Given a PDA

$$P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$$

We construct a CFG with variables

$$\text{var}(G) = \{A_{pq} \mid p, q \in Q\}$$

and start variable

$$S = A_{q_0 q_{accept}}$$

With rules

1° Single start state

2° Stack empty before accepting

3° Each transition `push` or `pop`, but not both

A new start $q_s \to q_{s'}$ with $\varepsilon, \varepsilon \to \$$, and for any $q \in F$, we have $\varepsilon, a \to \varepsilon$ back to $q$, $\forall a \in \Sigma$. Then from any $q \in F$, we do $\varepsilon, \$ \to \varepsilon$ to $q_a$

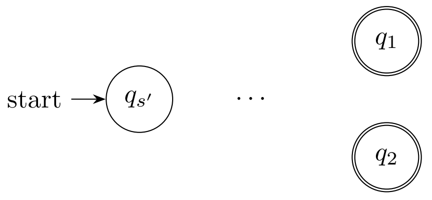
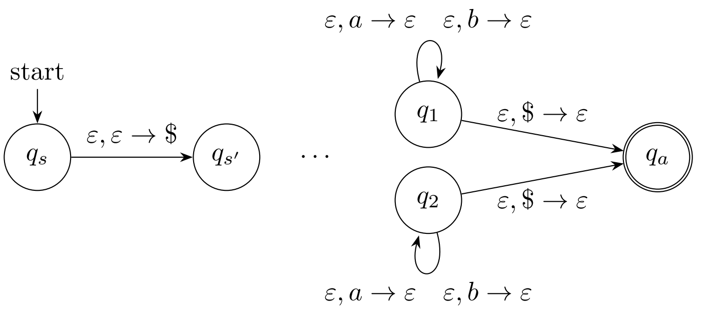


Figure 2.13: PDA with single accept state and empty stack before accepting

The new one will become


start
$q_s$
$\varepsilon, \varepsilon \to \$$
$q_{s'}$
$\cdots$
$\varepsilon, a \to \varepsilon \quad \varepsilon, b \to \varepsilon$
$q_1$
$\varepsilon, \$ \to \varepsilon$
$q_a$
$q_2$
$\varepsilon, \$ \to \varepsilon$
$\varepsilon, a \to \varepsilon \quad \varepsilon, b \to \varepsilon$

These is not enough to ensure condition (3°), we can do some modifications:

- To have each transition either `push` or `pop` (but not both), replace

  $$q_1 \xrightarrow{a,\, a \to b} q_2$$

  with the pair

  $$q_1 \xrightarrow{a,\, a \to \varepsilon} q_3, \qquad q_3 \xrightarrow{\varepsilon,\, \varepsilon \to b} q_2.$$

- Likewise, replace

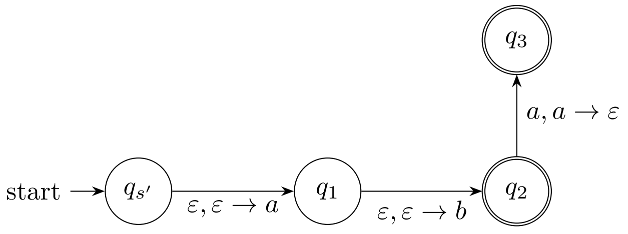
  $$q_1 \xrightarrow{a,\, \varepsilon \to \varepsilon} q_2$$

  with

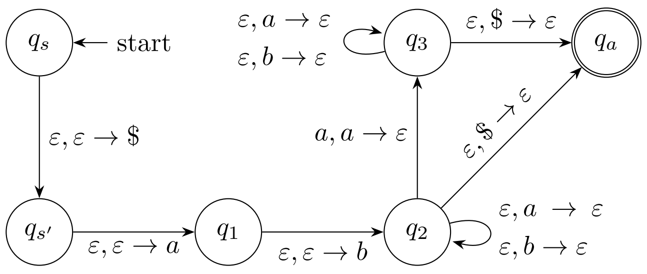
  $$q_1 \xrightarrow{a,\, \varepsilon \to X} q_3, \qquad q_3 \xrightarrow{\varepsilon,\, X \to \varepsilon} q_2,$$

  where $X$ is a fresh stack marker introduced for this simulation.

For another example, consider the PDA


$q_3$
$a, a \to \varepsilon$
start
$q_{s'}$
$\varepsilon, \varepsilon \to a$
$q_1$
$\varepsilon, \varepsilon \to b$
$q_2$

After the modification, we have


$q_s$
start
$\varepsilon, a \to \varepsilon$
$\varepsilon, b \to \varepsilon$
$q_3$
$\varepsilon, \$ \to \varepsilon$
$q_a$
$\varepsilon, \varepsilon \to \$$
$a, a \to \varepsilon$
$\varepsilon, \$ \to \varepsilon$
$q_{s'}$
$\varepsilon, \varepsilon \to a$
$q_1$
$\varepsilon, \varepsilon \to b$
$q_2$
$\varepsilon, a \to \varepsilon$
$\varepsilon, b \to \varepsilon$

The new PDA will accept the string $a$ but the original PDA rejects it. Hence, we need to modify something else:

- A new start $q_s \to q_{s'}$ with $\varepsilon, \varepsilon \to \$$
- A new state $q_{\text{pop}}$ that have $\varepsilon, a \to \varepsilon$ back to $q_{\text{pop}}$, $\forall a$.
- For $q \in F$, add a transition $\varepsilon, \varepsilon \to \varepsilon$ from $q$ to $q_{\text{pop}}$
- Add a new accept state $q_a$ and a transition $\varepsilon, \$ \to \varepsilon$ from $q_{\text{pop}}$ to $q_a$

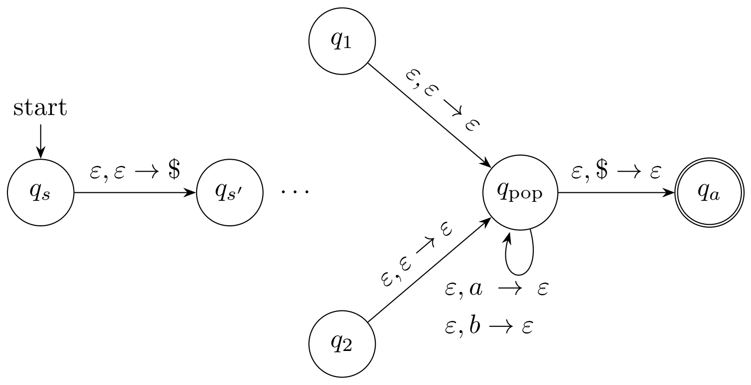

start
$q_s$
$\varepsilon, \varepsilon \to \$$
$q_{s'}$
$\cdots$
$q_1$
$\varepsilon, \varepsilon \to \varepsilon$
$q_2$
$\varepsilon, \varepsilon \to \varepsilon$
$q_{\text{pop}}$
$\varepsilon, a \to \varepsilon$
$\varepsilon, b \to \varepsilon$
$\varepsilon, \$ \to \varepsilon$
$q_a$

Figure 2.14: PDA with single accept state and empty stack before accepting

## 2.4 Deterministic Pushdown Automata

## Lecture 7

PDA is non-deterministic in general. However, there is a special class of PDA called Deterministic Pushdown Automata (DPDA). From Ch.1 we know 2025-11-03

$$\text{DFA} \equiv \text{NFA}$$

but

$$\text{DPDA} \neq \text{PDA} \implies \text{CFL} \neq \text{DCFL}$$

**Definition 2.4.1** (Deterministic Pushdown Automaton (DPDA)). A deterministic pushdown automaton (DPDA) is a 6-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$$

where

- $Q$: States
- $\Sigma$: Input alphabet
- $\Gamma$: Stack alphabet
- $\delta$: Transition function
$$Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to (Q \times \Gamma_\varepsilon) \cup \{\emptyset\}$$
- $q_0 \in Q$: Start state
- $F \subset Q$: Set of accepting states

To build a DPDA, we first look at the different between PDA and DPDA.

**As previously seen.** For PDA,

$$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$$

**Note.** In DPDA, for $\forall q \in Q, a \in \Sigma, x, \gamma \in \Gamma$, at most and at least one of the following is true:

$$\delta(q, a, x) = (p, \gamma), \quad \delta(q, a, \varepsilon) = (p, \gamma), \quad \delta(q, \varepsilon, x) = (p, \gamma), \quad \delta(q, \varepsilon, \varepsilon) = (p, \gamma)$$

the rest must be $\emptyset$.

### 2.4.1 Acceptance, Rejection of DPDA

The Rejection of DPDA is similar to PDA, which should only happen when

- Not end at an accept state after the last symbol.
- DPDA fails to read the input
  1. pop an empty stack
  2. Endless $\varepsilon$-transition

**Example.** $L = \{0^n 1^n \mid n \geq 0\}$

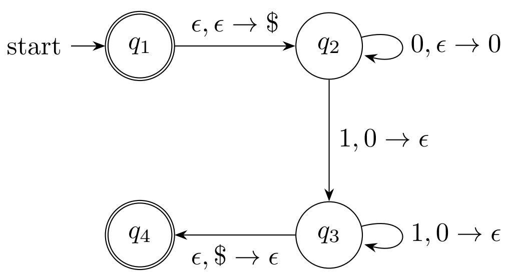


Figure 2.15: DPDA for $L = \{0^n 1^n \mid n \geq 0\}$

The Transition function is defined as follows:

| | 0 | | | 1 | | | $\epsilon$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | \$ | $\epsilon$ | 0 | \$ | $\epsilon$ | 0 | \$ | $\epsilon$ |
| $q_1$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $(q_2, \$)$ |
| $q_2$ | $\emptyset$ | $\emptyset$ | $(q_2, 0)$ | $(q_3, \epsilon)$ | $q_r$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| $q_3$ | $q_r$ | $\emptyset$ | $\emptyset$ | $(q_3, \epsilon)$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $(q_4, \epsilon)$ | $\emptyset$ |
| $q_4$ | $q_r$ | $q_r$ | $\emptyset$ | $q_r$ | $q_r$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| $q_r$ | $q_r$ | $q_r$ | $\emptyset$ | $q_r$ | $q_r$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |

To find this transition table, for instance,

- consider the state $q_1$:

$$\delta(q_1, \varepsilon, \varepsilon) = (q_2, \$)$$

  then we can implies that

$$\delta(q_1, a, \gamma) = \delta(q_1, a, \varepsilon) = \delta(q_1, \varepsilon, \gamma) = \emptyset, \quad \forall a \in \Sigma = \{0, 1\},\ \gamma \in \Gamma = \{0, \$\}$$

- consider the state $q_2$:

$$\delta(q_2, 1, 0) = (q_3, \varepsilon)$$

  then we can implies that

| | 0 | | | 1 | | | $\epsilon$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | \$ | $\epsilon$ | 0 | \$ | $\epsilon$ | 0 | \$ | $\epsilon$ |
| $q_2$ | $\emptyset$ | $\emptyset$ | $(q_2, 0)$ | $(q_3, \epsilon)$ | $\neq \emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |

  due to

$$\delta(q_2, 1, \varepsilon) = \delta(q_2, \varepsilon, 0) = \delta(q_2, \varepsilon, \varepsilon) = \emptyset$$

  Formally we have

$$\delta(q_2, 1, \$) = (q_r, \varepsilon)$$

For the string 011, the computation of the DPDA is as follows:

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

Follow the graph,

$$\delta(q_4, 1, \epsilon) \text{ and } \delta(q_4, \epsilon, \epsilon) = \emptyset$$

hence, the DPDA rejects 011.

# Chapter 3

# The Church-Turing Thesis

## Lecture 8

## 3.1 Turing Machines

2025-11-10

**As previously seen.** To discuss the computability of problems. We need a more powerful model. We already have seen

- Finite Automata (FA): with limited memory (states)
- Pushdown Automata (PDA): unlimited memory with LIFO structure (stack)

We now introduce a new computational model called **Turing Machine** (TM), which has an unlimited tape as memory.

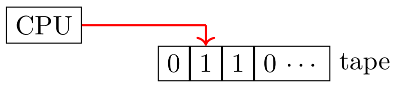


Figure 3.1: Illustration of a Turing Machine

A Turing Machine consists of these properties different from FA and PDA:

- write/read tape
- head that can move left/right on the tape
- unlimited tape length
- reject/accept take immediate effect
- machine can **never** halt

**Example.**

$$B = \{w\#w \mid w \in \{0,1\}^*\}$$

**Remark.** We can proof that this language is not CFL by pumping lemma for CFL.

**Notation.** $\sqcup$ is the blank symbol on the tape.

**Idea.** Zig-zag to the corresponding places on the two sides of the # and determine whether they match.

- Scan to check if ther is a #.
- Check $w$ and $w$ if they match.

$$\begin{array}{ccccccccccccccc}
\overset{\vee}{0} & 1 & 1 & 0 & 0 & 0 & \# & 0 & 1 & 1 & 0 & 0 & 0 & \sqcup \\
x & \overset{\vee}{1} & 1 & 0 & 0 & 0 & \# & 0 & 1 & 1 & 0 & 0 & 0 & \sqcup \\
x & 1 & 1 & 0 & 0 & 0 & \# & \overset{\vee}{x} & 1 & 1 & 0 & 0 & 0 & \sqcup
\end{array}$$

Figure 3.2: Illustration of algorithm

**Definition 3.1.1** (Turing Machine (TM)). A Turing Machine is a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

where

- $Q$: States
- $\Sigma$: Input alphabet, where $\sqcup \notin \Sigma$
- $\Gamma$: Tape alphabet, where $\Sigma \subset \Gamma$ and $\sqcup \in \Gamma$
- $\delta$: Transition function
$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$
- $q_0 \in Q$: Start state
- $q_{\text{accept}} \in Q$
- $q_{\text{reject}} \in Q,\ q_{\text{reject}} \neq q_{\text{accept}}$

The input
$$w = w_1 w_2 \cdots w_n \in \Sigma^*$$
will be put in the position $1, 2, \cdots, n$ of the tape, and the rest of the tape is filled with $\sqcup$.

**Example.**
$$L = \{0^{2^n} \mid n \geq 0\}$$

**Idea.** Cross off every second, and check if the remaining is even (except the last one).

$$0\underline{0}0\underline{0}$$
$$0\underline{0}$$
$$0$$

The procedure should be:

1° left → right, make remark on every second 0

2° if step 1° left with only one unmarked 0, accept

3° if step 1° left with odd #0 left, reject

4° move head to the leftmost

5° go to step 1°

The definition of the machine is

$$\begin{aligned} Q &= \{q_0, q_1, q_2, q_3, q_4, q_5, q_{\text{accept}}, q_{\text{reject}}\} \\ \Sigma &= \{0\} \\ \Gamma &= \{0, x, \sqcup\} \end{aligned}$$

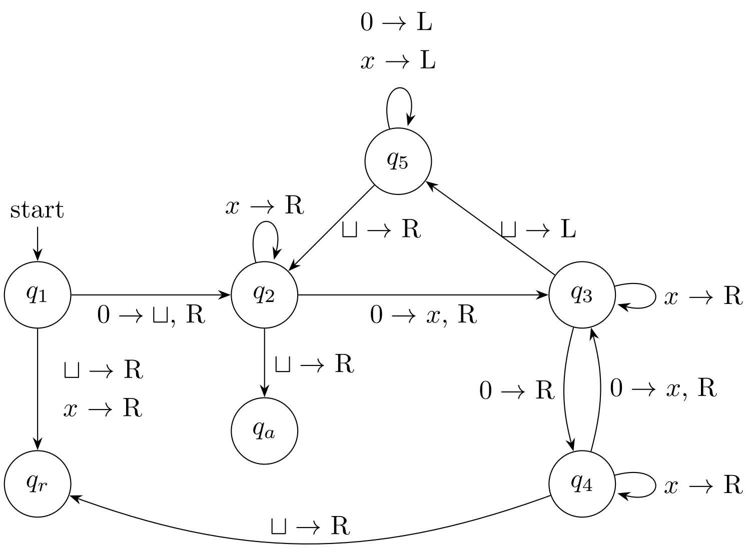


Figure 3.3: TM for $L = \{0^{2^n} \mid n \geq 0\}$

**Notation.**

$$0 \to R \equiv 0 \to 0, R$$

Consider the input 0000:

| | | | | |
|---|---|---|---|---|
| $q_1 0000$ | $\sqcup q_2 000$ | $\sqcup x q_3 00$ | $\sqcup x 0 q_4 0$ | $\sqcup x 0 x q_3$ |
| $\sqcup x 0 q_5 x$ | $\sqcup x q_5 0 x$ | $\sqcup q_5 x 0 x$ | $q_5 \sqcup x 0 x$ | $\sqcup q_2 x 0 x$ |
| $\sqcup x q_2 0 x$ | $\sqcup x x q_3 x$ | $\sqcup x x x q_3 \sqcup$ | $\sqcup x x q_5 x$ | $\sqcup x q_5 x x$ |
| $\sqcup q_5 x x x$ | $q_5 \sqcup x x x$ | $\sqcup q_2 x x x$ | $\sqcup x q_2 x x$ | $\sqcup x x q_2 x$ |
| $\sqcup x x x q_2$ | $\sqcup x x x \sqcup q_a$ | | | |

The transition function table is as follows:

| | 0 | x | ⊔ |
|---|---|---|---|
| $q_1$ | $q_2, \sqcup, R$ | $q_{reject}, x, R$ | $q_{reject}, \sqcup, R$ |
| $q_2$ | $q_3, x, R$ | $q_2, x, R$ | $q_{accept}, \sqcup, R$ |
| ⋮ | | | |

**Note.** There is no need for transition for $q_{\text{accept}}$ and $q_{\text{reject}}$ since the machine halts when it enters these states.

**Idea.** We can get the design idea of Turing Machine

- $q_1$ : mark the start by $\sqcup$
  - first element must be 0, otherwise, reject
  - Using $\sqcup$, so the start is known
- $q_2 \to q_3$: handle initial 00
- $q_3 \to q_4 \to q_3$: sequentially $00 \to 0x$
  - If not pairs (e.g., 0x0x0x), fails
  - This is the place of checking if # of remained zeros is even
- $q_3 \to q_5 \to q_2$ back to beginning
- first First 0 (or $\sqcup$) is considered the single final 0

$$q_2 \to \cdots \to q_2 \to \cdots \to q_{accept}$$

check if a single 0 is left in the string.

### 3.1.1 Configuration of Turing Machine

**Definition 3.1.2 (current configuration).** The current configuration of a Turing Machine is represented as

$$uqv$$

where

- $u \in \Gamma^*$: the string on the left of the head
- $q \in Q$: the current state
- $v \in \Gamma^*$: the string on the right of the head

The head is reading the first symbol of $v$. If $v = \epsilon$, then the head is reading a blank symbol $\sqcup$.

**Definition 3.1.3.** $a, b, c \in \Gamma$, $u, v \in \Gamma^*$, $q_i, q_j \in Q$ then the transition from configuration

- If $\delta(q_i, b) = (q_j, c, L)$, then
$$uaq_ibv \vdash uq_jacv$$
- If $\delta(q_i, a) = (q_j, b, L)$, then
$$uaq_ibv \vdash uacq_jv$$

### 3.1.2 Turing Recognizable and Turing Decidable Languages

**Definition 3.1.4** (Turing Recognizable)**.** A language $L$ is Turing recognizable if some Turing Machine $M$ recognizes it.

For a Turing Machine there are three possible outcomes:

- Accept the input by entering $q_{\text{accept}}$
- Reject the input by entering $q_{\text{reject}}$
- Loop forever without halting

A language is very difficult to difficult to decide if the TM loops forever on some inputs. We now define a more restricted type of model, called **Decider**.

**Definition 3.1.5** (Turing Decidable)**.** A language $L$ is Turing decidable if some Turing Machine $M$ decides it.

We will discuss more about Decidability in later chapters (Ch.4).

### 3.1.3 Example of Turing Machine

**Example.** $L = \{w\#w \mid w \in \{0,1\}^*\}$

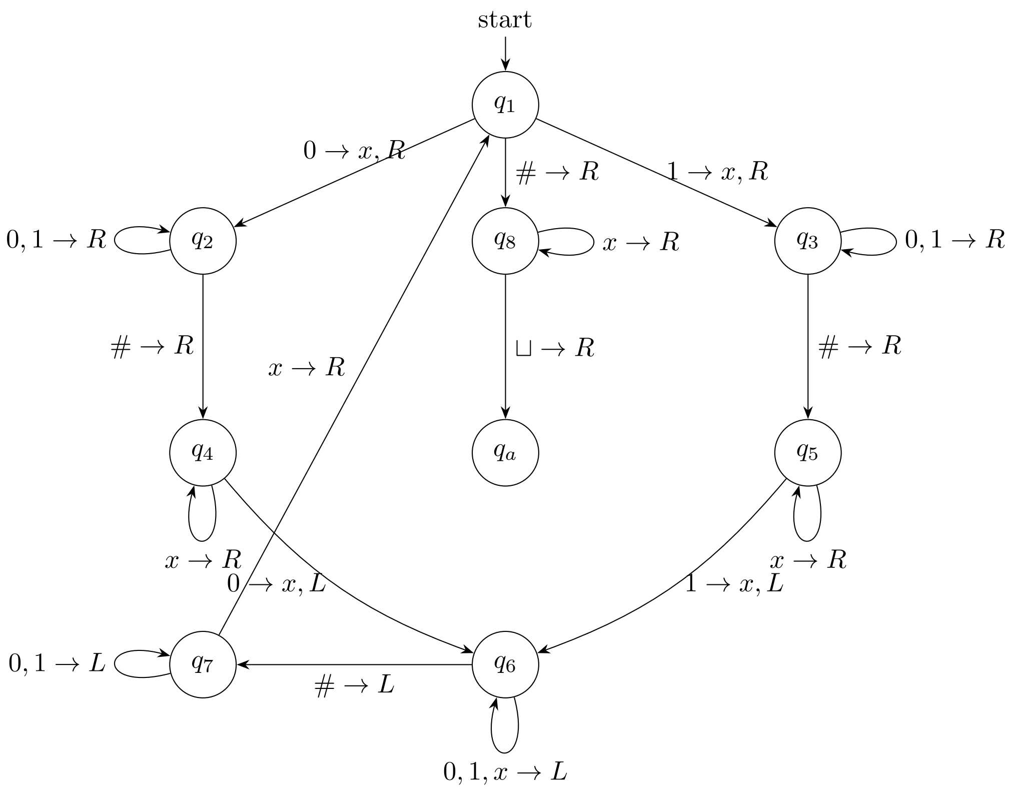


Figure 3.4: Turing Machine of $L = \{w\#w \mid w \in \{0,1\}^*\}$

**Remark.** Links to $q_r$ are not shown

Simulate 01#01

$$\begin{array}{llll} q_1 01\#01 & xq_2 1\#01 & x1q_2\#01 & x1\#q_4 01 \\ x1q_6\#x1 & xq_7 1\#x1 & q_7 x1\#x1 & xq_1 1\#x1 \\ xxq_3\#x1 & xx\#q_5 x1 & xx\#xq_5 1 & xx\#q_6 xx \\ xxq_6\#xx & xq_7 x\#xx & xxq_1\#xx & xx\#q_8 xx \\ xx\#xxq_8 \sqcup & xx\#xx \sqcup q_a & & \end{array}$$

**Idea.** The diagram:

$$q_1 \to q_2 \to q_4 \to q_6$$

check 0 at the same position of the two strings

$$q_1 \to q_3 \to q_5 \to q_6$$

check 1 at the same position of the two strings

**Example.** $C = \{a^i b^j c^k \mid i \times j = k,\ i, j, k \geq 1\}$

**Idea.** The procedure should be:

1° check if the input is $a^+ b^+ c^+$

2° back to the leftmost $a$

3° fix an $a$, for each $b$, cross off a $c$

4° store $b$ back, cancel one $a$, repeat step 3

- Step 1 can be done by a DFA (as DFA is a special case of TM).
- Step 2 can be done by moving left until $\sqcup$ is reached.
- Step 3 is similar to previous examples.

**Example.** $E = \{\#x_1\#x_2 \cdots \#x_l \mid x_i \in \{0,1\}^*,\ x_i \neq x_j\}$

**Idea.** Sequentially compare every pairs

$$\begin{array}{l} x_1x_2, x_1x_3, \ldots, x_1x_l \\ x_2x_3, \ldots, x_2x_l \\ \vdots \\ x_{l-1}x_l \end{array}$$

For $x_i,\ x_j$, mark #'s strings by $\dot{\#}$ i.e.

$$\dot{\#}x_1\#x_2\dot{\#}x_3 : x_1 \text{ and } x_3 \text{ are compared}$$

We can copy $x_i$, $x_j$ to the right end of the tape and compare them there with the pattern of $w\#w$.