§1. Right-linear Grammars & NFAs

Def. let \( L(\text{REX}) \) be the class of all languages that can be described by regular expressions, \( L(\text{RLG}) \) those that can be generated by right-linear grammars, and \( L(\text{NFA}) \) (resp. \( L(\text{DFA}) \)) those that can be recognized by NFAs (resp. DFAs).

Our main aim in this chapter is to first show that \( L(\text{RLG}) = L(\text{NFA}) = L(\text{QAS-NFA}) = L(\text{REX}) \).

\[ L(DFA) \]

We have already seen that \( L(DFA) = L(NFA) \) because every DFA is a special NFA, so \( L(DFA) \subseteq L(NFA) \); and every NFA is equivalent to a DFA, so \( L(NFA) \subseteq L(DFA) \).

Prop. 1. Let \( M = \langle Q, \Sigma, \delta, q_0, A \rangle \) be an NFA. Then we can find an RLG \( G = \langle V, \Sigma, \{ S \}, P \rangle \) such that \( L(G) = L(M) \).

Proof: let \( V = \emptyset \) and \( S = q_0 \). The productions in \( P \) are constructed from \( M \) as follows:

For each \( B \in A(M) \) in \( M \), we get the production \( B \rightarrow \). Also each transition \( A \xrightarrow{} B \) or \( A \xrightarrow{} B \) in \( M \), we get the production \( A \rightarrow aB \) or resp. \( A \rightarrow B \).

From the construction of \( G \) we can see that: we \( L(M) \) \( \iff \) we can lead you from \( q_0 \) to an accepting state in \( M \) \( \iff \) there is a derivation of \( w \), from \( S \), in \( G \) \( \iff \) we \( L(G) \). So \( d(b) = 4 \).
Let $M$ be the NFA shown on the right. Then an equivalent RLG $G$ will be as shown below.

$\rightarrow A, \ A \rightarrow aB, \ B \rightarrow bB, \ B \rightarrow C, \ C \rightarrow aC, \ C \rightarrow bA, \ C \rightarrow \lambda$.

Let us consider the string $\varphi = abba$. In $M$, $\varphi$ is accepted as follows: $\rightarrow A \rightarrow B \rightarrow B \rightarrow B \rightarrow C \rightarrow C$.

In $G$, $\varphi$ can be derived from $A$ as follows:

$\rightarrow A \rightarrow aB \rightarrow abB \rightarrow abB \rightarrow abbc \rightarrow abbc \rightarrow abcC \rightarrow abbcC \rightarrow abbcC \rightarrow abba$.

Let $M = \rightarrow B \rightarrow A \rightarrow C$. Then an equiv.

RLG will be as shown below.

$\rightarrow B, \ B \rightarrow aA, \ A \rightarrow \lambda, \ A \rightarrow B, \ A \rightarrow bC, \ C \rightarrow \lambda, \ C \rightarrow bB, \ C \rightarrow aC$.

Notice that the number of productions in $G$ will be the number of transitions of $M$ plus the number of states in $A(N)$.

Prop 2. Let $G = \langle V, T, \{S\}, \varphi \rangle$ be an RLG. Then we can find an NFA $M = \langle Q, T, \Delta, q_0, A \rangle$ such that $L(M) = L(G)$.

Proof. Let $q_0 = S$. We will construct $Q, A$ and $\Delta$ from the productions of $G$ as follows. $Q$ will start out being $V$ - but more states will be added as we go along.
(a) For each production in G of the form B → λ, let B ∈ A(M).
(b) For each production of the form A → B in G, add the transition \( A \rightarrow B \) to M.
(c) For each production in G of the form \( A \rightarrow a_1 \ldots a_n B \), make \( n-1 \) new states and add the transitions \( A \rightarrow a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_n B \) to M.
(d) Finally, make a specially designated accepting state \( Z \) in A(M) and for each production in G of the form \( A \rightarrow a_1 a_2 \ldots a_k \), make \( k-1 \) new states & add the transitions \( A \rightarrow a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_k \rightarrow Z \) to M.

From the construction of M, we can see that \( L(M) = L(G) \) because each derivation in G of a string \( q \) will correspond to an acceptance track of \( q \) in M.

Ex. 2: Let G be the RLG shown below.

\[
S \rightarrow abS, \quad S \rightarrow A, \quad A \rightarrow baB, \quad A \rightarrow aA, \quad A \rightarrow \lambda,
\]

\[
B \rightarrow cC, \quad B \rightarrow ba, \quad C \rightarrow ac, \quad C \rightarrow bc
\]

Then an equivalent NFA M will be as shown below.

The derivation \( S \rightarrow abS \rightarrow ab\lambda A \rightarrow ab\lambda aA \rightarrow ab\lambda \lambda A \) in G corresponds to the acceptance track \( S \rightarrow 1 \rightarrow \lambda \rightarrow S \rightarrow A \rightarrow A \) in M.
Let \( M = \langle Q, \Gamma, \Delta, q_0, A \rangle \) be an NFA and \( \psi \) and \( \varphi \) be strings from \( \Gamma^* \). Recall that the reaching set \( R(\psi) \) is the set of all states that can be reached via \( \psi \) when starting at \( q_0 \).

We define the back-tracing set \( B(\psi) \) by \( B(\psi) = \{ q \in Q : \exists \eta \in \Gamma^+ \text{ such that } \eta \psi \eta \in L(M) \} \). The set \( \psi B(\psi) \) is the set of all states from which you can start and reach an accepting state via \( \psi \).

**Question 1**

Given an RLG, \( G \), how can we tell if it is ambiguous?

**Answer 1**

First convert the RLG, \( G \), into an equivalent NFA \( M \). Then find all the reaching sets and back-tracing sets of \( M \). \( G \) will be ambiguous \( \iff \) there is a reaching set \( R(\psi) \) and a back-tracing set \( B(\psi) \) such that \( |R(\psi) \cap B(\psi)| \geq 2 \). The string \( \psi \eta \psi \) will have at least two leftmost derivations.

**Question 2**

Given an ambiguous RLG, \( G \), how can we find an equivalent un-ambiguous RLG, \( G' \)?

**Answer 2**

First convert the RLG, \( G \), into an equivalent NFA. Then convert \( M \) into an equivalent DFA, \( M_0 \). Finally, convert \( M_0 \) into an equivalent RLG, \( G' \). Then \( G' \) will be an un-ambiguous RLG which is equivalent to \( G \). The ambiguity was removed in passing from the NFA to the DFA.
Ex. 3  
(a) Determine whether or not the RLG \( G \) below is ambiguous.  
\( S \to bA, \quad S \to bB, \quad A \to aB, \quad A \to b, \quad A \to ab, \quad B \to aB, \quad B \to a. \)  
(b) If \( G \) is ambiguous, find an equivalent un-ambiguous RLG, \( G' \).

(a) \[
\begin{align*}
& b \\
& S \quad A \\
& b \quad B \\
& b \quad Z
\end{align*}
\]

Reaching Sets:
\[
\begin{array}{c}
\{S\} \\
b \quad \{A,B\} \\
b \quad \{Z\} \\
b \quad \emptyset \\
b \quad \emptyset \\
b \quad \emptyset
\end{array}
\]

Back-Tracking Sets:
\[
\begin{array}{c}
\{A,B\} \\
\{A,B\} \\
\{S\} \\
\emptyset \\
\emptyset \\
\emptyset
\end{array}
\]

If we take \( \varphi = b \) and \( \psi = aa \), then we see that \( R(\varphi) = \{A,B\} \) and \( B(\psi) = \{A,B\} \). Since
\[
|R(\varphi) \cap B(\psi)| = |\{A,B\} \cap \{A,B\}| = |\{A,B\}| = 2 > 2,
\]

it follows that \( G \) is ambiguous. The string \( \varphi \psi = baa \) will have at least 2 leftmost derivations in \( G \). Indeed, we have:
\[
\begin{align*}
& S \Rightarrow bA \Rightarrow baB \Rightarrow baa \quad \text{(first leftmost derivation)} \\
& S \Rightarrow bB \Rightarrow baB \Rightarrow baa \quad \text{(2nd leftmost derivation)}
\end{align*}
\]
(b) We already found the reaching sets of M in part (a). So it is now easy to find the DFA, $M_0$.

Now all we have to do is to convert $M_0$ into an equivalent RLG, $G'$. To make this easier to understand, let $C = \{s\}$, $D = \{A, B\}$, $E = \emptyset$, $F = \{B, Z\}$ and $H = \{z\}$. Then $G'$ is

$\to C, \ C \to aE, \ C \to bD, \ D \to aF, \ D \to bH, \ E \to aE, \ E \to bE, \ F \to bE, \ F \to aF, \ H \to aE, \ H \to bE, \ F \to \lambda, \ H \to \lambda$,

and we are done. Note that the string $yyp = baa$ has only one leftmost derivation in $G'$ because there is only one path from $\{s\} = C$ to $\{B, Z\} = F$ via $baa$.

§2. Regular expressions & OAS-NFAs

Definition. An OAS-NFA is an NFA with only one accepting state which is different from the initial state.

It is easy to see that any NFA is equivalent to an OAS-NFA. All we have to do is to create a single new accepting state and add \( \lambda \)-transitions from the old accepting states to the single new one.
Prop. 3 If \( E \) is a regular expression, then we can find an OAS-NFA \( M \) such that \( L(M) = L(E) \).

Proof: Recall that a regular expression over \( \{a_1, \ldots, a_n\} \) is defined recursively as follows:
(a) \( a_1, \ldots, a_n \) and \( \emptyset \) are regular expressions.
(b) If \( E_1 \) and \( E_2 \) are regular expressions then so are \( (E_1 \cup E_2) \), \( (E_1 E_2) \), and \( (E_1^*) \).

To find an OAS-NFA for a reg. expr. \( E \), it will suffice to show how to make OAS-NFAs for the reg. expr. in (a) and given OAS-NFAs for \( E_1 \) & \( E_2 \) to show how to make OAS-NFAs for \( (E_1 \cup E_2) \), \( (E_1 E_2) \) and \( (E_1^*) \).

(a) OAS-NFA for \( a_i \):

\[
\begin{array}{c}
& \circ & \rightarrow & \circ \\
\end{array}
\]

OAS-NFA for \( \lambda \):

\[
\begin{array}{c}
& \circ & \rightarrow & \circ \\
\end{array}
\]

OAS-NFA for \( \emptyset \):

\[
\begin{array}{c}
& \circ & \rightarrow & \circ \\
\end{array}
\]

(b) OAS-NFA for \( (E_1 \cup E_2) \):

\[
\begin{array}{c}
& \circ & \rightarrow & \circ \\
\end{array}
\]

OAS-NFA for \( (E_1 E_2) \):

\[
\begin{array}{c}
& \circ & \rightarrow & \circ \\
\end{array}
\]

OAS-NFA for \( (E_1^*) \):

\[
\begin{array}{c}
& \circ & \rightarrow & \circ \\
\end{array}
\]

NFA for \( E_1 \)

NFA for \( E_2 \)
Ex. 4 Find an OAS-NFA M such that \( L(M) = a^* (bc + ca)^* \)

\[
\begin{align*}
\lambda & : \rightarrow O \rightarrow a \rightarrow O \\
bc & : \rightarrow O \rightarrow b \rightarrow \lambda \rightarrow c \rightarrow O \\
ca & : \rightarrow O \rightarrow c \rightarrow \lambda \rightarrow a \rightarrow O \\
\lambda & : \rightarrow O \rightarrow \lambda \\
(bc + ca)^* & : \rightarrow O
\end{align*}
\]

Ans:

M:

Ex. 5 Find an OAS-NFA M such that \( L(M) = (\emptyset^*) \)

Ans: \( \emptyset = \rightarrow O \rightarrow O \)

(\( \emptyset^* \): \rightarrow O)

Def: A generalized finite acceptor (GFA) is defined in the same manner as an NFA except that instead of having "\( a \)" or "\( \lambda \)" transitions, we can have any regular expression serve as a transition.

Ex. 6 Below is an example of a GFA

\[
\begin{align*}
A & \rightarrow ab^* \rightarrow a + bb \\
C & \rightarrow c^* (a + b) \\
B & \rightarrow ba^* b + cb^* \\
D & \rightarrow b^* + ab
\end{align*}
\]
Proposition 4: Let M be an NFA. Then we can find a regular expression E such that \( L(M) = E \).

Sketch of Proof: First we convert M into an OAS-NFA \( M_1 \). Then we view \( M_1 \) as a GFA and remove the other states, besides the initial state & accepting state, one at a time to get a sequence of equivalent, simpler & simpler GFAs. At the end, we will get a GFA as shown below with only two states.

![Diagram of two states and transitions](image)

Then \( L(M) = R_1 R_2 (R_4 + R_3 R_1 R_2)^* \).

Exercise: Find a regular expression for the language recognized by the NFA \( M \) below.

![Diagram of an NFA](image)

Answer: This NFA is an OAS-NFA, so we don't have to convert it into an OAS-NFA. To view \( M \) as a GFA, just add underlines under each transition.

![Diagram of a GFA](image)
Ex. 7

Eliminate D

\[ M_3 \rightarrow \begin{array}{cc}
A & B \\
\rightarrow & \rightarrow \\
b & ac \\
\delta^a & \delta^* \\
bca & c \\
\end{array} \]

Eliminate B

\[ M_4 \rightarrow \begin{array}{cc}
A & E \\
\rightarrow & \rightarrow \\
bc^*a & cac \\
bac & \delta^* \\
bca + \delta^* \\
\end{array} \]

So \( R_1 = b \lambda \delta^* a = bc^* a \), \( R_2 = bac \), \( R_3 = bc^* a + c \lambda \delta^* a = (b+c) \delta^* a \)

Hence \( L(M) = R_1^* R_2 \cdot (R_4 + R_3 R_1^* R_2)^* \)

\[ = (bc^*)^* (bac) \cdot (cac + \delta^* a) \cdot (bc^*)^* (bac) \]

Now from Propositions 1, 2, 3 & 4, we can deduce our first main result.

**Theorem 5 (Main theorem for Regular languages)**

\[ L(RLG) = L(NFA) = L(DFA) = L(DS-NFA) = L(REX) \]

**Closure Theorems for regular languages**

We will now explore some closure results about regular languages.

**Theorem 6. (Closure theorem for Regular languages)**

If \( L_1 \) & \( L_2 \) are regular languages, then so are also the following

(a) \( L_1 \cup L_2 \)
(b) \( L_1 \cdot L_2 \)
(c) \( L_1^* \)
(d) \( L_1^c \)
(e) \( L_2 \)
(f) \( L_1 L_2 \)
(g) \( L_1 - L_2 \)

**Proof:** Since \( L_1 \) & \( L_2 \) are regular languages, then by definition, we can find regular expressions \( E_1 \) & \( E_2 \) such that \( L(E_1) = L_1 \) & \( L(E_2) = L_2 \)

(a) Now \( (E_1 + E_2) \) is a regular expression which describes \( L_1 L_2 \). So \( L_1 L_2 \) is a regular language.
(b) \((E_1 E_2)\) is a regular expression which describes \(L_1 L_2\). So \(L_1 L_2\) is a regular language.

(c) \((E_1^*)\) is a regular expression which describes \(L_1^*\). So \(L_1^*\) is a regular language.

(d) Since \(L_1\) is a regular language, it follows from Theorem 5, the main theorem on reg. lang., that we can find an OAS-NFA \(M_1\) such that \(L(M_1) = L_1\). Now if we make the initial state of \(M_1\) into the accepting, make the accept state of \(M_1\) into the initial state, and reverse the direction of each transition, we will get an OAS-NFA \(M_1^R\) with \(L(M_1^R) = L_1^R\). So by Theorem 5, \(L_1^R\) is a regular language.

(e) First convert \(E_1\) into an equivalent NFA \(M_1\). Then convert \(M_1\) into an equivalent DFA, \(M_0\). Now form the complement machine \(M_0^C\) by switching the non-accepting states into accepting states & vice versa. Then convert \(M_0^C\) into an equiv. regular expression \(E^C\). We will then have \(L(E^C) = L(M_0^C) = L(M_0^C) = L(M_0^C) = L(E_1^C) = L_1^C\). So \(E^C\) will be a regular expression which describes \(L_1^C\). Hence \(L_1^C\) is a regular language.

(f) We know that \((L_1 \cap L_2)^c = L_1^c \cup L_2^c\) by De Morgan's Law. So \(L_1 L_2^c = (L_1 \cap L_2)^c = (L_1^c \cup L_2)^c\). But \(L_1^c \cup L_2^c\) are regular lang. by part (e). So \((L_1^c \cup L_2^c)\) is regular by part (a). Thus \(L_1 L_2^c = (L_1^c \cup L_2^c)^c\) will be regular by part (e).

(g) \(L_1 - L_2 = L_1 L_2^c\). By part (e), \(L_1^c\) is regular, so by part (f) \(L_1 - L_2 = L_1 L_2^c\) will a regular language.
Another closure result about regular languages.

**Def.** Let $T_1$ and $T_2$ be alphabets. We say that the function $h: T_1^* \rightarrow T_2^*$ is a homomorphism if $h(\alpha \beta) = h(\alpha) h(\beta)$ for any $\alpha, \beta \in T_1^*$.

**Fact 7.** If $h: T_1^* \rightarrow T_2^*$ is a homomorphism, then we must have $h(\lambda) = \lambda$.

**Proof:** We know that $h(\alpha \beta) = h(\alpha) h(\beta)$ for any $\alpha, \beta \in T_1^*$. So in particular since $\lambda \tau = \tau \lambda$, we get $h(\lambda) = h(\lambda \lambda) = h(\lambda) h(\lambda)$. From this it follows that $h(\lambda) = \lambda$ — because if $h(\lambda) \neq \lambda$, then we cannot have $h(\lambda) = h(\lambda) h(\lambda)$.

It can also be shown that $h$ is completely determined by $h(c_1), h(c_2), \ldots, h(c_n)$ where $T_1 = \{c_1, c_2, \ldots, c_n\}$. Now if $E$ is a regular expression over $T_1$, then $h(E)$ can be defined as the same expression $E$ except that each character $c_i$ is replaced by $h(c_i)$. For example, let $T_1 = \{a, b, c\}$ & $T_2 = \{0, 1\}$, put $h(a) = 10$, $h(b) = 11$ & $h(c) = 00$. If $E = a^* (b + ac)^*$, then we will have that $h(E) = (10)^* (11 + 1000)^*$.

**Prop. 8.** If $L$ is a regular language on $T_1$ and $h: T_1^* \rightarrow T_2^*$ is a homomorphism, then $h[L] = \{h(\alpha) : \alpha \in L\}$ is also a regular language.

**Proof:** If $E$ is a reg. exp. for $L$, then $h(E)$ is a reg. exp. for $h[L]$. 
Let $L = \{ w \in \{0,1\}^* : w$ contains the string 00 $\}$. Find a regular expression which describes $L$ and a regular expression which describes $L^c = \{ w : w$ does not contain 00 $\}$.

(a) $E = (0+1)^*$. 00. $(0+1)^*$ is a regular expression which describes $L$.

(b) First we convert $E$ into an equiv. NFA, $M$.

(c) Then we convert $M$ into an equiv. DFA, $M_D$.

Let $D = \{ A \}$, $E = \{ A, B \}$, $F = \{ A, B, C \}$ and $G = \{ A, C \}$.

Then $M_D$ will be

We can also minimize the DFA $M_D$ to get $(M_D)_{min}$ because $G$ is indistinguishable from $F$.

But we don't need to do this.

(d) Now switch the accepting & non-accepting states to get $(M_D)^c$.

(e) Then find the regular expression which $(M_D)$ recognizes. Since F & G are non-accepting states, we only need to look at the NFA. So convert this NFA into an OSA-NFA.

(f) Now get rid of $E$ to obtain the GFA.

(g) Final Answer is: $E^c = (1+0)^* (2+0)$. 
Ex. 9. Let $L = \{y \in \{0,1\}^* : y$ contains the string 101$\}$. Find regular expressions which describe $L$ and $L^c$.

(a) $(0+1)^*101(0+1)^*$ is a regular expression which describes $L$.

(b) To find a regular expression which describes $L^c$, we first find a DFA $M$ which recognizes $L$.

$$M = \begin{array}{c}
A & \xrightarrow{0} & B & \xrightarrow{1} & C & \xrightarrow{0} & D
\end{array}$$

We did this because it takes too many steps to go through an NFA.

(c) Now $M^c = \begin{array}{c}
A & \xrightarrow{0} & B & \xrightarrow{1} & C & \xrightarrow{0} & D
\end{array}$

So all we have to do now is to find the language which $M^c$ recognizes (as a regular expression).

(d) We can ignore $D$ and convert the rest of $M^c$ into an OAS-NFA.

$$M_1 = \begin{array}{c}
A & \xrightarrow{0} & B & \xrightarrow{1} & C & \xrightarrow{\lambda} & E
\end{array}$$

$$M_2 = \begin{array}{c}
A & \xrightarrow{0} & B & \xrightarrow{1} & \lambda & \xrightarrow{0+0} & Z
\end{array}$$

$$M_3 = \begin{array}{c}
\lambda & \xrightarrow{0+1(0+1)^*} & \lambda & \xrightarrow{0+1(0+1)^*} & Z
\end{array}$$

Note that $L^c = \{y \in \{0,1\}^* : y$ does not contain 101 as a substring$\}$. 

$$E^c = (0+1(0+1)^*101(0+1)^*)^* = (0+1(0+1)^*)^*(\lambda + (0+1(0+1)^*)^*101(0+1)^*)^*.$$
Def: A decision problem is a problem with unknown parameters which can take denumerably many values, such that for all values of the parameters, we have a yes or no answer.

Ex. 10 (Primality Decision Problem). The problem is:
Is $n$ a prime number? Here $n$ is the unknown parameter and for each value of $n$, the answer is either yes or no. The values the parameter $n$ can take are $\{0, 1, 2, 3, \ldots \} = \mathbb{N}$.

Def: A decision problem is algorithmically decidable if there is a fixed algorithm which takes as input the possible values of the unknown parameters and produces the correct answer.

Ex. 11 The Primality Decision Problem is algorithmically decidable because we can find an algorithm which can decide whether or not $n$ is prime. Just take $n$ as input and try to see if any integer between 1 and $\sqrt{n}$ divides $n$. If none of the integers from 2 to $\sqrt{n}$ divides $n$, then $n$ is prime. Otherwise $n$ is not prime.

Theorem 9 (Decidability Theorem for Reg. Lang.)
Let $L_1$ and $L_2$ be regular languages. Then it is algorithmically decidable whether or not:
(a) $L_1 = \emptyset$ (b) $L_1 \subseteq L_2$ (c) $L_1 = L_2$ (d) $L_1$ is infinite.
Before we prove Theorem 9, let us first point out that the parameters are a pair of regular languages, \( L_1 \) & \( L_2 \). There are only denumerably many values of these two parameters because the number of regular languages is denumerable. We need two lemmas to help us in Theorem 9.

**Def.** An NFA is said to be \( \lambda \)-free if there are no \( \lambda \)-transitions that can take you from one state to a different state.

**Lemma 10:** Let \( M \) be a \( \lambda \)-free NFA with \( N \) states. Then \( L(\lambda) \neq \emptyset \) \( \iff \) \( M \) accepts a string \( y_0 \) of length \( \leq N - 1 \).

**Proof:** (\( \iff \)) Suppose \( M \) accepts a string \( y_0 \) of length \( \leq N - 1 \). Then \( L(\lambda) \neq \emptyset \) because \( y_0 \in L(\lambda) \).

(\( \iff \)) Now suppose \( L(\lambda) \neq \emptyset \). Then we can find a string \( y_1 \) such that \( M \) accepts \( y_1 \). Consider any acceptance track of \( y_1 \). If we remove all the loops in this acceptance track of \( y_1 \), we will end up with an acceptance path of a string \( y_0 \).

Since the length of the longest possible path in a graph with \( N \) states is \( N - 1 \), we must have \( 1 + 1 \leq N - 1 \). So \( M \) accepts a string \( y_0 \) of length \( \leq N - 1 \).
Lemma 11: Let \( M \) be a \( \lambda \)-free NFA with \( N \) states. Then \( L(M) \) is infinite \( \iff \) \( M \) accepts a string \( q_0 \) with \( N \leq |q_0| \leq 2N-1 \).

Proof: (\( \Leftarrow \)) Suppose \( M \) accepts a string \( q_0 \) with \( N \leq |q_0| \leq 2N-1 \). Now consider any acceptance track of \( q_0 \) in \( M \). Since it takes \( |q_0|+1 \) states to process \( q_0 \) and \( |q_0| \geq N \), this acceptance track must have a loop as shown below. Let \( k = |q_0| \) and \( q_0 = c_1 c_2 \ldots c_k \).

\[
\begin{array}{c}
c_1 \rightarrow c_2 \rightarrow \cdots \rightarrow c_k \rightarrow c_{k+1} \rightarrow \cdots \rightarrow c_1 \rightarrow q_0 \\
\end{array}
\]

If we let \( x = c_1 c_2 \ldots c_i \), \( y = c_i+1 \ldots c_j \) and \( z = c_{j+1} \ldots c_k \), then we can see that for each \( r \geq 0 \), \( xy^r z \) will be accepted by \( M \). Since \( |y| = j-i \geq 1 \), it follows that all these strings \( xy^r z \) will be different from each other. So \( L(M) \) will be infinite.

(\( \Rightarrow \)) Suppose \( L(M) \) is infinite. Then \( M \) must accept a string \( q_0 \) with \( |q_0| > N \) — because the total number of strings with length \( \leq N-1 \) is finite. Now if \( |q_0| \leq 2N-1 \), then we take \( q_0 \) to be \( q_0 \), and \( M \) will accept a string \( q_0 \) with \( N \leq |q_0| \leq 2N-1 \). If \( |q_0| > 2N-1 \), then consider any acceptance track of \( q_0 \). If we delete the loops in this acceptance track of \( q_0 \), one at a time, we will eventually get a string \( q_0 \) which is accepted by \( M \) and which satisfies \( N \leq |q_0| \leq 2N-1 \). So \( M \) will accept a string \( q_0 \) with \( N \leq |q_0| \leq 2N-1 \).
Proof of Theorem 9

(a) Since $L_1$ is a regular language, it follows from Theorem 5 that we can find a DFA (or NFA) $M_1$ such that $L(M_1) = L_1$. Now from Lemma 10, we know that $L(M_1) \neq \emptyset \iff M_1$ accepts a string $q_0$ of length $\leq N-1$. So check all paths of length $\leq N-1$ from $q_0$ in $M$. If one of these paths leads to an accepting state, then $L(M_1) = L_1 \neq \emptyset$. If none of these paths lead to an accepting state, then $L_1 = \emptyset$. So we have an algorithm which can decide whether or not $L_1 = \emptyset$.

(b) First observe that $L_1 \subseteq L_2 \iff L_1 - L_2 = \emptyset$. Since $L_1 - L_2$ is a regular language by Theorem 6(f), it follows that we can use the algorithm in part (a) to check whether or not $L_1 - L_2 = \emptyset$. So we again have an algorithm to decide whether or not $L_1 \subseteq L_2$.

(c) Observe that $L_1 = L_2 \iff L_1 \subseteq L_2 \& L_2 \subseteq L_1$. So use the algorithm in part (b) twice to check if both $L_1 \subseteq L_2 \& L_2 \subseteq L_1$. So this gives us an algorithm to check whether or not $L_1 = L_2$.

(d) First find a $\lambda$-free NFA $M_1$ such that $L(M_1) = L_1$. Now $L(M_1)$ is infinite $\iff M_1$ accepts a string $q_0$ with $N \leq |q_0| \leq 2N-1$. So check all possible acceptance tracks in $M$ with lengths between $N$ and $2N-1$. If one track works, then $L(M_1) = L_1$ will be infinite. If none of the tracks works, then $L(M_1) = L_1$ will not be infinite.
§5. Non-regular languages & their properties

Prop. 12: The set of all regular languages on \( \Sigma = \{a, b, \lambda, 0, \vdash, \ast, (, )\} \) is countable and consequently non-regular languages exist.

Proof: We know that a regular language on \( \Sigma = \{a, b\} \) is one that can be described by a regular expression on \( \Sigma = \{a, b\} \). Now a reg. expr. on \( \Sigma = \{a, b\} \) is just a string based on the alphabet \( \{a, b, \lambda, 0, \vdash, \ast, (, )\} = T \). Since the set of all strings based on \( T \) is countable, it follows that \( \text{REX} \), the set of all regular expressions on \( \{a, b\} \) is also countable - because \( \text{REX} \subseteq T^* \). Since we only have a countable number of reg. expr., the total no. of regular languages must also be countable.

Finally, the set of all languages on \( \Sigma = \{a, b\} \) is uncountable, so there must exist languages that are not regular. These languages are naturally called non-regular.

We will now give several concrete examples of non-regular languages.

Ex. 1: \( L_1 = \{a^k b^k : k \geq 0\} \) is a non-regular language.

Proof: Suppose \( L_1 \) is regular. Then we can find a \( \lambda \)-free NFA such that \( L(M) = L_1 \), by Theorem 5. Let \( N = \text{number of states in } M \), and consider the string \( a^N b^N \). Since \( a^N b^N \notin L_1 \), \( M \) will accept
$a^N b^N$. Also since it takes $N+1$ states to process $a^N$, any acceptance track of $a^N b^N$ must have a loop as shown below.

Now if we skip the loop, we will see that $M_i$ accepts the string $a^i a^{N-i} b^N = a^{N-(i-i)} b^N$. Since $i < j$, $a^{N-(j-i)} b^N \notin L_j$. So we have a contradiction because $L(M_i) = L_j$ and $M_i$ accepted $a^{N-(i-i)} b^N$. Hence $L_j$ cannot be regular. So $L_j$ is non-regular.

Ex. 2 $L_2 = \{a^k b^m : k \leq m^2\}$ is a non-regular language.

Proof: Suppose $L_2$ is regular. Then we can find a $\Sigma$-free NFA $M_2$ such that $L(M_2) = L_2$. Let $N =$ the number of states in $M_2$ and consider the string $a^N b^N$. Since $a^N b^N \in L_2$, $M_2$ will accept $a^{N-i} b^N$. Also since it takes $N+1$ states to process $a^N$, any acceptance track of $a^N b^N$ must contain a loop as shown below.
Now if we ride the loop twice, we will see that $M_2$ accepts the string $a^i a^{j-i} a^{j-i} a^{N-j} b^N = a^{N+6-i} b^N$. But $a^{N+6-i} b^N \notin L_2$ because $N+(j-i) \notin N$ since $i \leq j$. This contradicts the fact that $L(M_2) = L_2$. Hence $L_2$ cannot be regular. So $L_2$ is non-regular.

**Ex. 3** $L_3 = \{a^{k^2} : k \geq 0\}$ is a non-regular language.

Proof: Suppose $L_3$ is regular. Then we can find a $\lambda$-free NFA $M_3$ such that $L(M_3) = L_3$. Let $N = \text{the number of states in } M_3 \text{ and consider the string } a^{(N+1)2} = a^{N^2+2N+1}$. Since $a^{N^2+2N+1} \notin L_3$, it will be accepted by $M_3$. Also, since it takes $2N+2$ states to process the last $(2N+1)$ $a$'s, any acceptance track of $a^{N^2+2N+1}$ must have a loop as shown below.

![Diagram](attachment:image.png)

Now if we skip the loop, we will see that $M_3$ accepts the string $a^{N^2} a^i a^{2N+1-i} = a^{N^2+2N+1-(j-i)}$. But $N^2 < N^2+2N+1-(j-i) < N^2+2N+1 = (N+1)^2$, so it follows that $N^2+2N+1-(j-i)$ cannot be a perfect square, because it is strictly between two consecutive perfect squares. Hence $a^{N^2+2N+1-(j-i)} \notin L_3$. But this contradicts the fact that $L(M_3) = L_3$. Hence $L_3$ cannot be regular. So $L_3$ is a non-regular language.
We can summarize the idea in the last 3 examples in the following result.

Prop.12 (Pumping Lemma for Regular Languages)
Let \( L \) be any infinite regular language. Then we can find a positive integer \( N \) such that any \( w \in L \) can be decomposed as \( w = xyz \) with \( |xy| \leq N \) and \( |y| > 1 \) such that \( xy^rz \in L \) for each \( r \in \mathbb{N} = \{0, 1, 2, 3, \ldots \} \).

Proof: Since \( L \) is regular, we can find a \( Q \)-free NFA \( M \) such that \( L(M) = L \). Put \( N \) the number of states in \( M \). Now if \( w \) is any string in \( L \) with \(|w| \geq N\), put \( w_N = \) the string of the first \( N \) characters of \( w \). Since \( w \in L, M \) will accept \( w_N \). Also since \( M \) has only \( N \) states, and we need \( N+1 \) states to process \( w_N \), any acceptance track of \( w \) must contain a loop as shown below.

![Diagram](image)

Now \(|xy| \leq N\) because the loop will appear somewhere between \( q_0 \) & \( q_N \). Also \(|y| = j-i \geq 1\). If we ride the loop \( r \) times, we will see that \( M \) accepts \( xy^rz \). So \( xy^rz \in L \) for each \( r \in \mathbb{N} \).
Ex.4 Using the Pumping Lemma for Reg. lang. prove that \( L_4 = \{a^pb^{2p} : 2p \leq n \} \) is non-regular.

Proof: Suppose \( L_4 \) is regular. Then by the Pumping Lemma, we can find an \( N \geq 1 \) such that any \( w \in L_4 \) with \(|w| \geq N\) can be decomposed as \( w = xyz \) with \(|xy| \leq N \) & \(|y| \geq 1\) such that \( xy^iz \in L_4 \) for all \( i \in \mathbb{N} \).

Now consider the string \( w = a^N b^{2N} \in L_4 \). Since the first \( N \) characters of \( w \) are all \( a \)'s, we must have \( x = a^p \), \( y = a^q \) and \( z = a^{N-p-q} b^{2N} \) for some \( p \) & \( q \) with \( 0 \leq p \leq N-1 \) & \( q \geq 1 \). But by the Pumping Lemma \( xy^0z \in L_4 \). So \( a^p a^{N-p-q} b^{2N} = a^{N-q} b^{2N} \in L_4 \).

But this contradicts the fact that strings in \( L_4 \) have twice as many \( b \)'s as \( a \)'s. Hence \( L_4 \) must be non-regular.

We can also prove that certain languages are non-regular by using the Closure Theorem & previous results.

Ex.5 Show that \( L_5 = \{a^k b^m : k \neq m \} \) is non-regular.

Proof: Suppose \( L_5 \) is regular. Then \( a^* b^* - L_5 \) will also be regular by Theorem 6(g). But \( a^* b^* - L_5 = \{a^{k} b^{\min(k,m)} : k \leq 0 \} \) \( = \{a^{k} b^{k} : k \geq 0 \} = L_1 \), which is non-reg.

So this is a contradiction. Hence \( L_5 \) is non-regular.
We will now explore some of the properties of non-regular languages.

**Prop. 13:** If \( L \) is a non-regular language, then

(a) \( L^c \) is non-regular (b) \( L^R \) is non-regular

**Proof:** (a) Suppose \( L^c \) is regular. Then by the Closure Theorem (Theorem 6(e)), it follows that \( (L^c)^c \) would also be regular. But \( (L^c)^c = L \), so this contradicts the fact that \( L \) was non-regular. Hence \( L^c \) must also be non-regular.

(b) Suppose \( L^R \) is regular. Then by the Closure Theorem part (d), \( (L^R)^R = L \) would also be regular. But this contradicts the fact that \( L \) is non-regular. Hence \( L^R \) must also be regular.

**Ex6**

If \( L_1 \) & \( L_2 \) are non-regular languages, it is possible for (a) \( L_1 \cup L_2 \) to be regular, (b) \( L_1 \cdot L_2 \) to be regular, and (c) \( L_1^* \) to be regular.

(a) Take \( L_1 = \{a^{2k} : k \geq 0\} \) and \( L_2 = \{a^k b^m : k \neq m\} \). Then \( L_1 \) & \( L_2 \) are both non-regular but \( L_1 \cup L_2 = a^* b^* \) which is regular.

(b) Take \( L_1 = \{a^{2k} : k \geq 0\} \) and \( L_2 = \{a^m : m \text{ is not a square}\} \). Then \( L_1 \cdot L_2 = a^* a^* \) is regular because

\[
L_1 \cdot L_2 = \{a^0, a^1, a^2, a^3, \ldots \} \cdot \{a^2, a^3, a^4, a^5, \ldots \} = \{a^2, a^3, a^4, a^5, \ldots \}
\]

(c) Take \( L_1 = \{a^{2k} : k \geq 0\} \). Then \( L_1^* = a^* \) which is regular because \( a^* = \{a^k : k \geq 0\} \subseteq \{a^0, a^1, a^2, a^3, \ldots \}^* \subseteq \{a \}^* \)

END