Part 1: Regular languages

Theme: The notion of regular languages, and three different but equivalent models of computation for regular languages: Deterministic finite state automata, non-deterministic finite state automata and regular expressions.

1 Deterministic finite state automata

A deterministic finite state automaton (DFA) is a system $A = \langle \Sigma, Q, q_0, F, \delta \rangle$, where each component is as follows.

- $\Sigma$ is the alphabet.
- $Q$ is a finite set of states.
- $q_0 \in Q$ is the initial state.
- $F \subseteq Q$ is the set of accepting states.
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function.

Remark 1.1 A DFA $A = \langle \Sigma, Q, q_0, F, \delta \rangle$ can be visualised as a directed graph as follows.

- The vertices are elements of $Q$.
- There is an edge from $p$ to $p'$ labeled with $a$, if $\delta(p, a) = p'$.

On input word $w = a_1 \cdots a_n$, the run of $A$ on $w$ is the sequence:

$$p_0 \ p_1 \ p_2 \ \cdots \ a_n \ p_n,$$

where $p_0 = q_0$ and $\delta(p_i, a_{i+1}) = p_{i+1}$, for each $i = 0, \ldots, n-1$.

Sometimes we are interested in a run that does not start from the initial state. In that case, we can define the run of $A$ on $w$ starting from state $q$ as the sequence defined as above, but with condition $p_0 = q$. That is,

$$p_0 \ p_1 \ p_2 \ \cdots \ a_n \ p_n,$$

where $p_0 = q$ and $\delta(p_i, a_{i+1}) = p_{i+1}$, for each $i = 0, \ldots, n-1$.

A run is called an accepting run, if $p_0 = q_0$ and $q_n \in F$. We say that $A$ accepts $w$, if there is an accepting run of $A$ on $w$. The language of all words accepted by $A$ is denoted by $L(A)$.

A language $L$ is called a regular language, if there is a DFA $A$ such that $L(A) = L$.

Remark 1.2 Let $A = \langle \Sigma, Q, q_0, F, \delta \rangle$ be a DFA.

- The empty string $\epsilon$ is accepted by $A$ if and only if $q_0 \in F$.
- For every word $w$, there is exactly one run of $A$ on $w$.

Theorem 1.3 Regular languages are closed under boolean operations, i.e., intersection, union, and complementation. More formally, it can be stated as follows.

- For every DFA $A$, there is a DFA $B$ such that $L(B) = \Sigma^* - L(A)$. 

$\square$
• For every two DFA $A_1$ and $A_2$, there is a DFA $B$ such that $L(B) = L(A_1) \cap L(A_2)$.
• For every two DFA $A_1$ and $A_2$, there is a DFA $B$ such that $L(B) = L(A_1) \cup L(A_2)$.

Proof. (Closure under complement) Let $A = (\Sigma, Q, q_0, F, \delta)$ be a DFA. Consider the DFA $B = (\Sigma, Q, q_0, Q - F, \delta)$, that is, $B$ is exactly the same as $A$ with the difference only in the accepting states, where the accepting states in $A$ become non-accepting in $B$ and the non-accepting states in $A$ become accepting in $B$.

Obviously, for every word $w \in \Sigma^*$, the accepting run of $A$ on $w$ becomes non-accepting run of $B$ on $w$. Vice versa, the non-accepting run of $A$ on $w$ becomes accepting run of $B$ on $w$. Thus, $L(B) = \Sigma^* - L(A)$.

(Closure under intersection) Let $A_1 = (\Sigma, Q_1, q_{0,1}, F_1, \delta_1)$ and $A_2 = (\Sigma, Q_2, q_{0,2}, F_2, \delta_2)$ be DFA. Consider the following DFA $B = (\Sigma, Q, q_0, F, \delta)$, where:

- $Q = Q_1 \times Q_2$.
- The initial state $q_0$ is $(q_{0,1}, q_{0,2})$.
- $F = F_1 \times F_2$.
- The transition function $\delta$ is defined as follows. For every $(p_1, p_2) \in Q_1 \times Q_2$, for every $a \in \Sigma$,

$$\delta((p_1, p_2), a) = (\delta_1(p_1, a), \delta_2(p_2, a))$$

We will show that $L(B) = L(A_1) \cap L(A_2)$. First, we show that $L(B) \subseteq L(A_1) \cap L(A_2)$. Consider a word $w = a_1 \cdots a_n$, where each $a_i \in \Sigma$. Suppose $w \in L(B)$, and we denote its accepting run by:

$$(s_0, t_0) a_1 (s_1, t_1) a_2 (s_2, t_2) \cdots a_n (s_n, t_n).$$

By the definition of accepting run and the definition of $B$,

$$s_0 \ a_1 \ s_1 \ a_2 \ s_2 \ \cdots \ \ a_n \ s_n \ \text{and} \ \ t_0 \ a_1 \ t_1 \ a_2 \ t_2 \ \cdots \ \ a_n \ t_n$$

are accepting runs of $A_1$ and $A_2$ on $w$, respectively, and hence, $w \in L(A_1) \cap L(A_2)$.

Now, we show that $L(A_1) \cap L(A_2) \subseteq L(B)$. Consider a word $w = a_1 \cdots a_n$, where each $a_i \in \Sigma$. Suppose $w \in L(A_1) \cap L(A_2)$. We denote the accepting run of $A_1$ on $w$ by:

$$s_0 \ a_1 \ s_1 \ a_2 \ s_2 \ \cdots \ \ a_n \ s_n$$

and the accepting run of $A_2$ on $w$ by:

$$t_0 \ a_1 \ t_1 \ a_2 \ t_2 \ \cdots \ \ a_n \ t_n.$$

By the definition of accepting run and the definition of $B$,

$$(s_0, t_0) a_1 (s_1, t_1) a_2 (s_2, t_2) \cdots a_n (s_n, t_n).$$

is the accepting run of $B$ on $w$. Hence, $w \in L(B)$.

(Closure under union) Similar to the intersection case, except that the accepting states are the elements in $(Q_1 \times F_2) \cup (F_1 \times Q_2)$. ■
2 Nondeterministic finite state automata

A _nondeterministic finite state automaton_ (NFA) is a system \( \mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle \), where each component is as follows.

- \( \Sigma \) is the alphabet.
- \( Q \) is a finite set of _states_.
- \( q_0 \in Q \) is the _initial_ state.
- \( F \subseteq Q \) is the set of _accepting_ states.
- \( \delta \subseteq Q \times \Sigma \times Q \) is the _transition_ relation.

On input word \( w = a_1 \cdots a_n \), a run of \( \mathcal{A} \) on \( w \) is the sequence:

\[ q_0 \ a_1 \ q_1 \ a_2 \ q_2 \ \cdots \ a_n \ q_n, \]

where \( (q_i, a_{i+1}, q_{i+1}) \in \delta \), for each \( i = 0, \ldots, n - 1 \). It is called _accepting_ run, if \( q_n \in F \). We say that \( \mathcal{A} \) _accepts_ \( w \), if there is an accepting run of \( \mathcal{A} \) on \( w \). The language of all words accepted by \( \mathcal{A} \) is denoted by \( L(\mathcal{A}) \). A language \( L \) is an NFA language, if there is an NFA \( \mathcal{A} \) such that \( L = L(\mathcal{A}) \), in which, we say that the language \( L \) is accepted by \( \mathcal{A} \), or \( \mathcal{A} \) accepts the language \( L \).

**Remark 1.4** NFA languages are closed under intersection and union. More formally, it can be stated as follows.

- For every two NFA \( \mathcal{A}_1 \) and \( \mathcal{A}_2 \), there is an NFA \( \mathcal{A}' \) such that \( L(\mathcal{A}') = L(\mathcal{A}_1) \cap L(\mathcal{A}_2) \).
- For every two NFA \( \mathcal{A}_1 \) and \( \mathcal{A}_2 \), there is an NFA \( \mathcal{A}' \) such that \( L(\mathcal{A}') = L(\mathcal{A}_1) \cup L(\mathcal{A}_2) \).

The proof is similar to the one in Theorem 1.3.

Question: Why can we not conclude that NFA languages are closed under complementation directly from the definition of NFA? ■

**Theorem 1.5** For every NFA \( \mathcal{A} \), there is a DFA \( \mathcal{A}' \) such that \( L(\mathcal{A}) = L(\mathcal{A}') \).

**Proof.** Let \( \mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle \) be an NFA. Consider the following DFA \( \mathcal{A}' = \langle \Sigma, Q', q'_0, F', \delta' \rangle \).

- \( Q' = 2^Q \), i.e., the set of all subsets of \( Q \), including \( \emptyset \) and \( Q \).
- The initial state \( q'_0 \) is \( \{q_0\} \), i.e., the set that contains only a single element which is the initial state \( q_0 \) of \( \mathcal{A} \).
- \( F' \) consists of the subset \( S \subseteq Q \) where \( S \cap F \neq \emptyset \).
- The transition function \( \delta' : 2^Q \times \Sigma \rightarrow 2^Q \) is defined as follows.

\[
\delta'(S, a) = \{ p \mid \text{there is } q \in S \text{ such that } (q, a, p) \in \delta \}
\]

In other words, we define for every \( S \in 2^Q \) and \( a \in \Sigma \), we define \( \delta'(S, a) \) to be the set \( T \), where \( p \in T \) if and only if there is \( q \in S \) such that \( (q, a, p) \in \delta \).

By default, we define \( \delta' (\emptyset, a) = \emptyset \).

We have the following two claims.

*As in the case of DFA, we can define a run of \( \mathcal{A} \) on \( w \) _starting from state q_ as above, but starts from state \( q_0 \).
Claim 1 For every word \( w \in \Sigma^* \), where \( w = a_1 \cdots a_n \), if there is a run of \( A \) on \( w \):

\[
q_0 a_1 q_1 a_2 q_2 \cdots a_n q_n, \quad \text{where } q_0 \text{ is the initial state of } A,
\]

then the run of \( A' \) on \( w \) denoted by:

\[
S_0 a_1 S_1 a_2 S_2 \cdots a_n S_n, \quad \text{where } S_0 \text{ is the initial state of } A',
\]
is such that \( p_i \in S_i \), for each \( i = 0, 1, \ldots, n \).

**Proof.** (of claim) The proof is by induction on the length of \( w \), i.e., \( n \). The base case, \( n = 0 \), holds trivially, since by the definition of \( A' \), its initial state \( q'_0 \) is the set \( \{q_0\} \).

For the induction hypothesis, assume that the claim holds for words of length \( n \). For the induction step, let \( w = a_1 \cdots a_n a_{n+1} \), i.e., of length \( n + 1 \). Suppose there is a run of \( A \) on \( w \):

\[
q_0 a_1 q_1 a_2 q_2 \cdots a_n q_n a_{n+1} q_{n+1}
\]

Applying the induction hypothesis on the word \( a_1 \cdots a_n \), the run of \( A' \) on \( a_1 \cdots a_n \) is:

\[
S_0 a_1 S_1 a_2 S_2 \cdots a_n S_n,
\]
is such that \( q_i \in S_i \), for each \( i = 0, 1, \ldots, n \).

Let \( S_{n+1} = \delta'(S_n, a_{n+1}) \). By the definition of run, \( (q_n, a_{n+1}, q_{n+1}) \in \delta \). Since \( q_n \in S_n \), by definition of \( \delta' \), \( q_{n+1} \in S_{n+1} \). Thus, the run of \( A' \) on \( a_1 \cdots a_n a_{n+1} \) is:

\[
S_0 a_1 S_1 a_2 S_2 \cdots a_n S_n a_{n+1} S_{n+1},
\]
where \( p_i \in S_i \), for each \( i = 0, 1, \ldots, n + 1 \). \( \blacksquare \)

Claim 2 For every word \( w \in \Sigma^* \), where \( w = a_1 \cdots a_n \), if the run of \( A' \) on \( w \) is as follows:

\[
S_0 a_1 S_1 a_2 S_2 \cdots a_n S_n, \quad \text{where } S_0 \text{ is the initial state of } A'
\]
then for every \( q \in S_n \), there is a run of \( A \) on \( w \):

\[
q_0 a_1 q_1 a_2 q_2 \cdots a_n q_n, \quad \text{where } q_0 \text{ is the initial state of } A
\]
such that \( q_n = q \).

**Proof.** (of claim) The proof is very similar to the claim above, i.e., by induction on \( n \). The base case, \( n = 0 \), holds trivially, since by the definition of \( A' \), its initial state is the set \( \{q_0\} \).

For the induction hypothesis, assume that the claim holds for words of length \( n \). For the induction step, let \( w = a_1 \cdots a_n a_{n+1} \), i.e., of length \( n + 1 \). Suppose the run of \( A' \) on \( w \) is as follows.

\[
S_0 a_1 S_1 a_2 S_2 \cdots a_{n+1} S_{n+1}, \quad \text{where } S_0 \text{ is the initial state of } A'
\]
Let \( q \in S_{n+1} \). By the definition of run, \( S_{n+1} = \delta'(S_n, a_{n+1}) \). By the definition of \( \delta' \), if \( q \in S_{n+1} \), there is \( p \in S_n \) such that \( (p, a_{n+1}, q) \in \delta \).

Applying the induction hypothesis on the word \( a_1 \cdots a_n \) and the state \( p \), there is a run of \( A \) on \( a_1 \cdots a_n \):

\[
q_0 a_1 q_1 a_2 q_2 \cdots a_n q_n.
\]
where \( q_n = p \). Since \((p, a_{n+1}, q) \in \delta\), we extend the run to be:

\[
q_0 a_1 q_1 a_2 q_2 \cdots a_n q_{n+1} a_{n+1} q_{n+1},
\]

where \( q_{n+1} = q \). ■

Note that Claim 1 implies \( L(A) \subseteq L(A') \) and Claim 2 implies \( L(A') \subseteq L(A) \). ■

In view of Theorem 1.5, we can say that a language is regular if and only if it is accepted by an NFA.

**Corollary 1.6** NFA languages are closed under complement. That is, for every NFA \( A \), there is a DFA \( A' \) such that \( L(A') = \Sigma^* - L(A) \).

### 3 NFA with \( \epsilon \)-move

An NFA with \( \epsilon \)-move (\( \epsilon \)-NFA) is a system \( A = (\Sigma, Q, q_0, F, \delta) \), where each component is as follows.

- \( \Sigma \) is the alphabet.
- \( Q \) is a finite set of states.
- \( q_0 \in Q \) is the initial state.
- \( F \subseteq Q \) is the set of accepting states.
- \( \delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q \) is the transition relation.

Intuitively, \((p, \epsilon, q) \in \delta\) means that the automaton \( A \) can move from state \( p \) to state \( q \) without reading any input symbol.

On input word \( w = a_1 \cdots a_n \), a run of \( A \) on \( w \) is the sequence:

\[
q_0 b_1 q_1 b_2 q_2 \cdots b_m q_m,
\]

where

- each \( b_i \) is either \( \epsilon \) or a symbol from \( w \);
- each \((q_i, b_{i+1}, q_{i+1}) \in \delta\);
- \( b_1 \cdots b_m = a_1 \cdots a_n \).

The conditions for accepting run and acceptance of a word are defined similarly as in NFA.

**Theorem 1.7** For every \( \epsilon \)-NFA \( A \), there is an NFA \( A' \) such that \( L(A) = L(A') \).

**Theorem 1.8** Regular languages are closed under concatenation and Kleene star. More formally, it can be stated as follows.

- If \( L_1 \) and \( L_2 \) are regular languages, so is \( L_1 L_2 \).
- If \( L \) is a regular language, so is \( L^* \).
4 Pumping lemma

In the following, for a word \( w \) and an integer \( n \geq 0 \), \( w^n \) obtained by repeating \( w \) for \( n \) number of times, i.e., \( w \cdots w \). By default, we define \( w^0 = \varepsilon \).

Lemma 1.9 (pumping lemma) Let \( A = \langle \Sigma, Q, q_0, F, \delta \rangle \) be an NFA. Suppose there is a word \( x \in L(A) \) such that \( |x| \geq |Q| \). Then, the word \( x \) can be divided into three parts \( u, v, w \), i.e., \( x = uvw \), such that \(|v| \geq 1 \) and for every positive integer \( i \geq 0 \), \( uv^i w \in L(A) \).

Proof. Let \( x = a_1 \cdots a_n \) and \( x \in L(A) \), where \( n \geq |Q| \). Let the following be its accepting run:

\[ p_0 a_1 p_1 a_2 p_2 \cdots a_n p_n \]

Since \( n \geq |Q| \), there are \( 0 \leq i < j \leq n \) such that \( p_i = p_j \).

Let \( u = a_1 \cdots a_i \), \( v = a_{i+1} \cdots a_j \) and \( w = a_{j+1} \cdots a_n \). Then, for every integer \( i \geq 0 \), the following is an accepting run of \( A \) on \( uv^i w \):

\[ p_0 a_1 p_1 a_2 p_2 \cdots a_i p_i \underbrace{a_{i+1} p_{i+1} \cdots a_j p_j}_{\text{repeat } i \text{ times}} a_{j+1} p_{j+1} \cdots a_n p_n \]

\[ \blacksquare \]

Lemma 1.10 (more refined pumping lemma) Let \( A = \langle \Sigma, Q, q_0, F, \delta \rangle \) be an NFA, and let \( x, y, z \) be words such that \( xyz \in L(A) \) and \(|y| \geq |Q| \). Then, the word \( y \) can be divided into three parts \( u, v, w \), i.e., \( y = uvw \), such that \(|v| \geq 1 \) and for every positive integer \( i \geq 0 \), \( xuv^i wz \in L(A) \).

5 Regular expressions

In the following we fix an alphabet \( \Sigma \). Regular expressions (over \( \Sigma \)) are expressions built inductively as follows.

- \( \emptyset \) is a regular expression.
- \( a \) is a regular expression, for every symbol \( a \in \Sigma \).
- If \( e_1, e_2 \) are regular expressions, then so are \((e_1 \cdot e_2)\) and \((e_1 \cup e_2)\).
- If \( e \) is a regular expression, then so is \((e)^*\).

A regular expression \( e \) over \( \Sigma \) defines a language, denoted by \( L(e) \), over the same alphabet as follows.

- If \( e = \emptyset \), then \( L(e) = \emptyset \).
- If \( e = a \), where \( a \in \Sigma \), then \( L(e) = \{a\} \).
- If \( e \) is of the form \((e_1 \cdot e_2)\), where \( e_1 \) and \( e_2 \) are regular expressions, then \( L(e) = L(e_1) \cdot L(e_2) \).
- If \( e \) is of the form \((e_1 \cup e_2)\), where \( e_1 \) and \( e_2 \) are regular expressions, then \( L(e) = L(e_1) \cup L(e_2) \).
- If \( e \) is of the form \((e_1)^*\), where \( e_1 \) is a regular expression, then \( L(e) = L(e_1)^* \).

Usually, we omit writing \( \cdot \) in \((e_1 \cdot e_2)\), and instead, we simply write \((e_1 e_2)\). Also, when there is no ambiguity, we will omit writing the brackets and simply write \( e_1 e_2 \) and \( e_1^* \), instead of \((e_1 e_2)\) and \((e_1)^* \).

The following theorem states that the class of languages defined by regular expressions is exactly the class of regular languages.
Theorem 1.11  Regular expressions define precisely the class of regular languages. More formally,

- for every regular expression \( e \) over \( \Sigma \), \( L(e) \) is a regular language, and
- for every NFA \( A \), there is a regular expression \( e \) such that \( L(e) = L(A) \).

Proof. We first prove the first item. The proof is by induction on the regex \( e \). The base case is when \( e \) is either \( \emptyset \) or \( a \in \Sigma \).

- When \( e = \emptyset \), then \( L(e) = \emptyset \).
  One can easily construct an NFA \( A \) that accepts nothing.
- When \( e = a \), for some symbol \( a \in \Sigma \), then \( L(e) = \{a\} \).
  We can construct an NFA \( A \), that has only two states \( p \) and \( q \), \( p \) is the initial state and there is only one accepting state \( q \) and \( \delta \) contains only one transition \((p, a, q)\).

For the induction step, we will prove the case where \( e \) is either of the form \( \alpha \cdot \beta \), \( \alpha \cup \beta \) or \( \alpha^* \).

By the induction hypothesis, there are NFAs \( A_1 \) and \( A_2 \) that accept the languages \( L(\alpha) \) and \( L(\beta) \), respectively. By Remark 1.4 and Theorem 1.8, there are NFAs for all the languages \( L(\alpha \cdot \beta) \), \( L(\alpha \cup \beta) \) and \( L(\alpha^*) \).

We now prove the second item. Let \( A = (\Sigma, Q, q_0, F, \delta) \) be an NFA. Without loss of generality, we assume that \( Q = \{1, \ldots, n\} \). For every \( 1 \leq i, j \leq n \) and \( 0 \leq k \leq n \), define the language \( L(i, j, k) \) as follows.

\[
L(i, j, k) := \left\{ w \left| \begin{array}{l} \text{there is a run of } A \text{ on } w \text{ from state } i \text{ to state } j \text{ without passing any states } \geq k + 1 \end{array} \right. \right\}
\]

That is, if \( w \in L(i, j, k) \), there is a run of \( A \) on \( w \) from state \( i \) to \( j \) without passing through the states \( k + 1, \ldots, n \).

We will first prove the following claim.

Claim 3  For every \( 1 \leq i, j \leq n \) and \( 0 \leq k \leq n \), define the language \( L(i, j, k) \), there is a regex \( e \) such that \( L(e) = L(i, j, k) \).

Proof. The proof is by induction on \( k \). The base case is \( k = 0 \). For every \( 1 \leq i, j \leq n \), consider the set of symbols \( \Gamma_{i,j} = \{a \mid (i, a, j) \in \delta\} \).

- If \( \Gamma_{i,j} = \emptyset \), then \( L(i, j, 0) = \emptyset \). The desired regex \( e \) is:
  \[
e = \begin{cases} \emptyset & \text{if } i \neq j \\ \emptyset^* & \text{if } i = j \end{cases}
  \]

- If \( \Gamma_{i,j} \neq \emptyset \), assume \( \Gamma_{i,j} = \{a_1, \ldots, a_t\} \). The desired regex \( e \) is:
  \[
e = \begin{cases} a_1 \cup \cdots \cup a_t & \text{if } i \neq j \\ a_1 \cup \cdots \cup a_t \cup \emptyset^* & \text{if } i = j \end{cases}
  \]

For the induction hypothesis, we assume that the claim holds for \( k \). For the induction step, we will prove it for \( k + 1 \). Note the following identity:

\[
L(i, j, k + 1) = L(i, j, k) \cup \left( L(i, k + 1, k) \cdot L(k + 1, k + 1, k) \cdot L(k + 1, j, k) \right)
\]
By the induction hypothesis, there are regexes that define each of $L(i, j, k), L(i, k + 1, k), L(k + 1, k + 1, k)$, and $L(k + 1, j, k)$. By the definition of regex, there is regex that define $L(i, j, k + 1)$. ■

To complete our proof, note that:\n\[
L(A) = \bigcup_{q_f \in F} L(q_0, q_f, n)
\]
By the claim above, for each $L(q_0, q_f, n)$, there is a regex that defines it. Thus, there is a regex for $L(A)$. ■

Combining what we have learnt so far, we obtain three different, but equivalent, characterisations of regular languages, as stated below.

**Corollary 1.12** Let $L$ be a language. Then, the following are equivalent.

- $L$ is accepted by a DFA.
- $L$ is accepted by an NFA.
- $L$ is defined by a regular expression.

**Remark 1.13** The term regular expressions are commonly abbreviated as regex. In most literatures and websites, the term “regex” are used more often than “regular expression.” Due to its widespread applications, many modern programming languages now include libraries for regex. The following are some of them.

- Java: [https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html](https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html)
- Python: [https://docs.python.org/2/library/re.html](https://docs.python.org/2/library/re.html)

**Appendix: Concatenation and Kleene star**

For two words $u$ and $v$, $u \cdot v$ denotes the word obtained by concatenating $v$ at the end of $u$. ($u \cdot v$ reads: $u$ concatenates with $v$.) By default, $u \cdot \epsilon = \epsilon \cdot u = u$. We will usually omit $\cdot$ and simply write $uv$ instead of $u \cdot v$.

In the following, let $L_1, L_2$ and $L$ be languages. We define the following operators.

\[
L_1 \cdot L_2 := \{uv \mid u \in L_1 \text{ and } v \in L_2\} \quad \text{(Concatenation)}
\]

\[
L^n := \{u_1 \cdots u_n \mid \text{each } u_i \in L\}
\]

\[
L^* := \bigcup_{n \geq 0} L^n \quad \text{(Kleene star)}
\]

As before, we usually write $L_1L_2$ to denote $L_1 \cdot L_2$, and $L_1L_2$ reads as $L_1$ concatenates with $L_2$.

Note that by default, for any set $X \subseteq \Sigma^*$, $X^0 = \{\epsilon\}$. Thus, $\emptyset^* = \{\epsilon\}$. 

8