11.1 Regular Languages

11.1.1 Definition and Examples

Definition 11.1 A Language \( L \) is said to be regular if there exists a DFA \( M \) such that \( L(M) = L \).

We can group together all those languages for which there are DFAs that recognize them into one single class of languages, called regular languages. Examples of regular languages are:

- The language \( L = \{ w \in \{a, b\}^* | w \text{ has two consecutive } b's \} \) is regular since there is a DFA that recognizes it.
- The language \( L = \{ w \in \{a, b\}^* | w \text{ has an even number of } a's \text{ and an even number of } b's \} \) is regular since there is a DFA that recognizes it.

Are all languages regular? To answer this question we can try to find a DFA for \( L = \{ w \in \{a, b\}^* | w = a^n b^n a^{n-1} b^{n-1} \ldots b \text{ where } n \in \mathbb{N} \} \). This is the language of those strings with \( n \) a’s followed by \( n \) b’s. The number of a’s must be equal to the number of b’s, and there is no limit as to the length of the string.

We can start by building a DFA that accepts those strings in \( L \) with length 6 or less, i.e. at most 3 a’s and 3 b’s (see figure 11.1.1).

In this machine some transitions are not shown, like that from the start state on input b. Since we know that the language does not contain strings that start with a b, it is assumed that this transition goes to an error state, from which there is no way out. A DFA where some transitions are not given is called an incompletely specified DFA.
There are problems for which we require a complete DFA, in that case we need to transform the incompletely specified DFA into a complete DFA, by adding a non-final state and making all missing transitions go into this state (see figure 11.1.1).

If we now wanted to build a DFA that would accept \( L \), for all possible values of \( n \), it would require an infinite number of states. This claim is done informally, and is meant only as an aid to understand the intuitive idea of why there are languages that are not regular. Later we will show that in fact \( L \) is not regular, and therefore, no DFA can exist that accepts \( L \).

### 11.1.2 Regular Operations

The following operations are applied to languages. They can be applied to any class of languages, not only to regular languages, even though they are called regular operations.

**Definition 11.2** Assume that \( L_1 \) and \( L_2 \) are languages. The following three operations are called regular operations.

1. \( L_1 \cup L_2 = \{ w \in \Sigma^* | w \in L_1 \text{ or } w \in L_2 \} \) (union of two languages)

2. \( L_1 L_2 = \{ w \in \Sigma^* | w = xy, x \in L_1, y \in L_2 \} \) (concatenation of two languages). A language can be concatenated with itself several times, and this situation is denoted by \( L L = L^2 \), \( LLL = L^3 \), and so on.

3. \( L_1^* = \{ \epsilon \} \cup L_1 \cup L_1^2 \cup L_1^3 \cup L_1^4 \cup \ldots \) (Kleene star)

**Union** The union operation is treated exactly the same as the union of two sets, since languages are sets.
**Concatenation** The concatenation of two strings $x$ and $y$, is a new string formed by the string $x$ followed by the string $y$. The concatenation of a language $L_1$ to another language $L_2$ is formed by all possible strings where its first part belongs to $L_1$ and its second part belongs to $L_2$. For example, if $L_1 = \{aa, abb, aba\}$, and $L_2 = \{bb, baa, bab\}$, the concatenation of $L_1$ to $L_2$ is

$$L_1L_2 = \{aabb, aabaa, aabab, abbbb, abbaa, abbbab, ababb, ababaa, ababab\}$$

the substrings that come from $L_2$ are underlined for clarity.

**Kleene Star** This operator is a unary operator, i.e. it is applied only to a single set. In the case where the language is $L = \{a, b\}$, the kleene star of $L$ is the language containing the empty string, all the elements of $L$ (strings with one symbol), all the elements of $L^2$ (in this case all strings with two symbols) and so on.

$$L^* = \{\epsilon, a, b, aa, ab, ba, bb, aab, aba, abb, baa, bab, bbb, \ldots\}$$

That is why we use $\Sigma^*$ to represent the set of those strings that can be written using symbols from $\Sigma$. From now on, whenever we will refer to $\Sigma^*$ we refer to $\{a, b\}^*$ unless we explicitly say otherwise.

11.1.3 Closure Properties

**Definition 11.3** A set $A$ is said to be closed under an operation $\circ$ if when the operation $\circ$ is applied to elements from $A$, the result is also an element from $A$.

**Example.**

- The set of natural numbers (positive integers) is closed under the addition ($+$) operation. There is no way that when adding two positive integers, the result is something other than a positive integer.
- The set of natural numbers is not closed under the subtraction ($-$) operator. As an example, take 5 and 8 which are natural numbers, but $5-8=-3$ is no longer a natural number.
- The set of rational numbers (fractions) is closed under the addition operator ($+$) (see the section about proof methods).
- The set of real numbers is not closed under the unary $\sqrt{}$ operator. For example, given the real number $-1$, $\sqrt{-1}$ is not a real number.
Theorem 11.4 Regular languages are closed under the following operations:

1. Union
2. Concatenation
3. Kleene Star

Since regular languages are closed under union, given two regular languages $L_1$ and $L_2$, then $L_1 \cup L_2$ is also a regular language. Therefore there is a DFA $M$ that accepts $L_1 \cup L_2$. This is a good method for creating DFAs that accept more complex languages. Before giving a proof for the first part of theorem 11.4, we will show how to build a DFA that accepts the union of two regular languages.

**Example.** Let $L_1 = \{ w \in \Sigma^* | w \text{ has two consecutive } b \text{'s} \}$, and let $L_2 = \{ w \in \Sigma^* | w \text{ starts and ends with a } b \}$. The transition diagram of machines $M_1$ and $M_2$, such that $L_1 = L(M_1)$ and $L_2 = L(M_2)$ are shown below.

$M_1$:

![Diagram for $M_1$]

$M_2$:

![Diagram for $M_2$]

We want to build a new machine $M$ that accepts the union of both languages, $L_1 \cup L_2$. Therefore, $M$ should accept those strings $w \in \Sigma^*$ that are in $L_1$ or in $L_2$. Since the computation of a DFA is done one symbol at a time, it is not possible to feed $w$ to $M_1$, and after it is done with its computation, feed it to $M_2$, i.e. we need to create **ONE SINGLE** DFA that accepts strings in $L_1 \cup L_2$. 

4
We will carry out the computation on both machines at the same time. Imagine that looking at the diagram of both machines, you put one finger on the current state of machine $M_1$ and another finger on the current state of machine $M_2$. Then you read one symbol and move both fingers to the state determined by the transition function of each machine.

Suppose that we want to process the string $babab$ through both machines simultaneously. Initially you would have each finger pointing to state 0 of each machine. Once you read symbol $b$ from the input string, the finger pointing to machine $M_1$ would now point to state 1, and the finger pointing to machine $M_2$ would now point to state 1. We could write down the entire computation using an idea similar to instantaneous configurations, $[(\text{finger 1, finger 2), unread string}]$. The entire trace of the computation on both machines is shown below.

\[
[(0, 0), babab] \rightarrow [(1, 1), abab] \\
\rightarrow [(0, 2), bab] \\
\rightarrow [(1, 1), ab] \\
\rightarrow [(0, 2), b] \\
\rightarrow [(1, 1), \epsilon]
\]

The string $babab$ is accepted because the computation ends with machine $M_1$ in state 1, and machine $M_2$ in state 1, which is an accepting state. Since $w \in L_2$, then $w \in L_1 \cup L_2$.

Notice that this computation can be seen as the computation of one single DFA, except that instead of using states in the instantaneous configuration, this computation uses pairs of states. This gives us the idea that we can build a new DFA that accepts $L_1 \cup L_2$ by using pairs of states from the original two DFAs. It is easy to see that we can create the transition table, by using the idea of one finger keeping track of the state of each machine. The transition table for our new states (pairs of states) is shown in table 11.1.3.

The start state of this new machine $M$ is $(0, 0)$, and the set of final states, is given by those pairs $(q_i, q_j)$ such that either $q_i$ or $q_j$ is a final state of its respective machine. Therefore, $F = \{(2, 0), (2, 1), (2, 2), (2, 3), (0, 1), (1, 1)\}$. Notice that the computation trace done above ends in state $(1, 1)$ which is an accepting state.

At this point we have the complete formal description of a new DFA $M$ that accepts $L_1 \cup L_2$. Figure 11.1.3 gives the transition diagram of $M$.

Several states are not reachable from the start state, i.e. there is no computation that can use them because they don’t have any transitions going into them. Those states can be safely removed from $M$. We will see later that there is a way to minimize a given DFA.
Figure 3: DFA $M$ accepting the union of $L_1$ and $L_2$. 
Table 1: Transition table for $M$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0)</td>
<td>(0,3)</td>
<td>(1,1)</td>
</tr>
<tr>
<td>(0,1)</td>
<td>(0,2)</td>
<td>(1,1)</td>
</tr>
<tr>
<td>(0,2)</td>
<td>(0,2)</td>
<td>(1,1)</td>
</tr>
<tr>
<td>(0,3)</td>
<td>(0,3)</td>
<td>(1,3)</td>
</tr>
<tr>
<td>(1,0)</td>
<td>(0,3)</td>
<td>(2,1)</td>
</tr>
<tr>
<td>(1,1)</td>
<td>(0,2)</td>
<td>(2,1)</td>
</tr>
<tr>
<td>(1,2)</td>
<td>(0,2)</td>
<td>(2,1)</td>
</tr>
<tr>
<td>(1,3)</td>
<td>(0,3)</td>
<td>(2,3)</td>
</tr>
<tr>
<td>(2,0)</td>
<td>(2,3)</td>
<td>(2,1)</td>
</tr>
<tr>
<td>(2,1)</td>
<td>(2,2)</td>
<td>(2,1)</td>
</tr>
<tr>
<td>(2,2)</td>
<td>(2,2)</td>
<td>(2,1)</td>
</tr>
<tr>
<td>(2,3)</td>
<td>(2,3)</td>
<td>(2,3)</td>
</tr>
</tbody>
</table>

The previous example describes how to build a machine $M$ that accepts the union of two regular languages, given the two DFAs that accept $L_1$ and $L_2$ respectively.

**Proof.** (by construction) Let $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ be two DFAs that accept languages $L_1$ and $L_2$ respectively. We will build a DFA $M = (Q, \Sigma, \delta, q_0, F)$ that accepts $L_1 \cup L_2$.

- $Q = \{(q_i, q_j) | q_i \in Q_1 \text{ and } q_j \in Q_2\}$. The states of our new machine $M$ are all possible pairs of states of the original two machines.
- For simplicity we will assume that the alphabet $\Sigma$ is the same for $L_1$ and $L_2$.
- The transition function is computed by using the idea of one finger on the current state of each machine. $\delta : Q \times \Sigma \rightarrow Q$, where $\delta((q_i, q_j), s) = (\delta_1(q_i, s), \delta_2(q_j, s))$ and $s \in \Sigma$.
- $q_0 = (q_{01}, q_{12})$
- $F = \{(q_i, q_j) | q_i \in F_1 \text{ or } q_j \in F_2\}$

Now we need to show that the construction is correct, i.e. that if a string $w$ is in $L_1 \cup L_2$ then $M$ accepts it, and that if $M$ accepts a string $w$ then $w \in L_1 \cup L_2$. It should be obvious from the construction that this is the case, so the formal proof is left as an exercise. \[\Box\]