Theorem (The Pumping Theorem). Let $A$ be a regular language. There exists $n \geq 1$ such that any string $x$ of $A$ where the length of which $|x| \geq n$ can be rewritten as $x = uvw$ such that $|uv| \leq n$, $|v| \geq 1$ and $uv^iw$ belongs to $A$ for all $i \geq 0$.

In a more compact form, the pumping theorem can be written as follows:

\[ \forall A \text{ regular languages } \exists n \geq 1 \text{ such that } \forall x \in A : |x| \geq n \exists x = uvw, \text{ such that } |uv| \leq n, |v| \geq 1 \forall i \geq 0 \text{ } uv^iw \in A. \]

The pumping theorem can be used in showing a language $L$ to be not regular using contradiction. First assume that $L$ is a regular language. The goal is to end in contradiction with this assumption by following the demands imposed on $L$ by the pumping theorem.

When using the theorem we have to remember that it works only in one direction. It can show that a language is not regular, but it cannot be used to show nonregularity of a language. For example, the language $I = \{c^ia^nb^n \mid i > 0 \land n \geq 0\} \cup (a^*b^*)$ is not regular, but all words in it may be partitioned in a way that satisfies the requirements of the theorem. Thus, it is not possible to use the theorem directly to prove that $I$ is not regular. In this case we have to use an indirect proof using the closure properties of regular languages. The answer to exercise 5 shows how this is done.

4. Problem: Pattern expressions are a generalisation of regular expression used e.g. in some text editing tools of UN*X*-type operating systems. In addition to the usual regular expression constructs, a pattern expression may contain string variables, inducing the constraint that any two appearances of the same variable must correspond to the same substring. Thus e.g. $aX^*Xa$ and $aX(a \cup b)^*YX(a \cup b)^*Ya$ are pattern expressions over the alphabet $\{a, b\}$. The first one of these describes the language $\{awb^nwa \mid w \in \{a, b\}^*, n \geq 0\}$. Pattern expressions are a proper generalisation of regular expressions, i.e. that pattern expressions can be used to describe also some nonregular languages.

Solution:
Th prove that pattern expressions are a proper generalization os regular expressions, we must find a pattern expression that defines a language that is not regular.

Consider the pattern expression $XX$. The corresponding language is $L = \{zz \mid z = \{a, b\}^*\}$. Assume that $L$ is regular. Select $x = a^nba^n \in L$. Now $|x| = 2n+2 > n$. According to the pumping theorem, we can rewrite $x$ as $x = uvw$, where $|uv| \leq n$ and $|v| \geq 1$. Now we have $u = a^{n-|v|-k}$, $v = a^{|v|}$ and $w = a^kba^n$, where $0 \leq k < n$. According to the theorem, for all $i \geq 0$ it should hold that $uv^iw \in L$. Still, $uv^0w = uw = a^{n-|v|}ba^n \notin L$, for it is not of the form $zz$, as $|v| \geq 1$. This is contradiction with the assumption that $L$ is a regular language. Therefore, $L$ is not a regular language.

A non-regular language was found that can be expressed using pattern expressions. Therefore, pattern expressions are a proper generalisation of regular expressions. □

5. Problem: Prove that the language $L = \{w \in \{a, b\}^* \mid w \text{ contains equally many } a\text{'s and } b\text{'s}\}$ is not regular, and design a context-free grammar generating it.

Solution:
It would be possible to use the pumping theorem directly to show that \( L \) is not regular. However, we will use a little more complex solution, because there is no other example in the course material that shows how “difficult” nonregular languages can be handled.

We define a language \( L' = L \cap L(a^*b^*) \). Suppose that \( L \) is regular. Then, \( L' \) has to be also regular since the class of regular languages is closed under intersection and \( L(a^*b^*) \) is regular (note that this condition does not hold to the other direction: \( L' \) may be regular even if \( L \) is not. For example, \( A \cap \emptyset = \emptyset \) for all languages \( A \)).

We note that \( L' = \{a^kb^k \mid k \geq 0\} \). Next, we examine the word \( w = a^n b^n \) where \( n \) is the language-dependant length-bound given by the pumping theorem (so \( |w| > n \)). Now we try to partition \( w \) in a way that satisfies the conditions of the theorem. Since \( |xy| \leq n \), the partition has to be of the form:

\[
\begin{align*}
x &= a^{n-i} \\
y &= a^i \\
z &= b^n,
\end{align*}
\]

where \( 0 < i \leq n \). However, now \( xz = a^{n-i}b^n \), so \( xz \notin L' \). Since \( w \) cannot be pumped, \( L' \) is not regular. However, this is a contradiction with our assumption that \( L \) is regular, so \( L \) may not be regular, either.

The following context-free grammar \( G \) defines \( L \): \( G = (V, \Sigma, P, S) \), where

\[
\begin{align*}
V &= \{S, T, a, b\}, \\
\Sigma &= \{a, b\},
\end{align*}
\]

\[
P = \{ S \to SS \mid aT \mid Ta \mid \varepsilon, \\
T \to ST \mid TS \mid b \}
\]

For example, the word \( aababb \in L \) may be derived as follows:

\[
\begin{align*}
S &\Rightarrow aT \\
&\Rightarrow aST \\
&\Rightarrow aabT \\
&\Rightarrow aabST \\
&\Rightarrow aabaTT \\
&\Rightarrow aababT \\
&\Rightarrow aababb
\end{align*}
\]

6. **Problem**: Design a context-free grammar describing the syntax of simple “programs” of the following form: a program consists of nested for loops, compound statements enclosed by begin-end pairs and elementary operations a. Thus, a “program” in this language looks something like this:

\[
a;
\text{for 3 times do}
\begin{align*}
&\text{begin} \\
&\text{for 5 times do a;} \\
&a; a
\end{align*}
\text{end.}
\]

For simplicity, you may assume that the loop counters are always integer constants in the range 0,..., 9.

**Solution**: The context-free grammars of programming languages are most often defined so that the alphabet consists of all syntactic elements (lexemes) that occur in the language. In this case numbers, a, and reserved words are lexemes. We divide the parsing of a program into two parts:

(a) The program text is transformed into a string of lexemes using a finite state automaton;
(b) The parse tree of the lexeme string is constructed.

The given grammar can be formalized in many ways, this is one possible interpretation:

$$G = (V, \Sigma, P, C)$$

$$V = \{C, S, N, \text{begin, do, end, for, times}, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ;, a\}$$

$$\Sigma = \{\text{begin, do, end, for, times}, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ;, a\}$$

Here the nonterminal $S$ denotes a statement, $C$ a compound statement, and $N$ a number. The rules of the grammar are defined as follows:

$$P = \{C \rightarrow S \mid S; C \mid \text{begin } C \text{ end} \mid \text{for } N \text{ times do } S \mid N \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\}$$

For example, the program in the problem text can be derived as follows:

$$S' \Rightarrow C$$

$$\Rightarrow S; C$$

$$\Rightarrow a; C$$

$$\Rightarrow a; S$$

$$\Rightarrow a; \text{for } N \text{ times do } S$$

$$\Rightarrow a; \text{for } 3 \text{ times do } S$$

$$\Rightarrow a; \text{for } 3 \text{ times do begin } C \text{ end}$$

$$\Rightarrow a; \text{for } 3 \text{ times do begin } S; C \text{ end}$$

$$\Rightarrow a; \text{for } 3 \text{ times do begin for } N \text{ times do } S; C \text{ end}$$

$$\Rightarrow a; \text{for } 3 \text{ times do begin for } 5 \text{ times do } S; C \text{ end}$$

$$\Rightarrow a; \text{for } 3 \text{ times do begin for } 5 \text{ times do } a; C \text{ end}$$

$$\Rightarrow a; \text{for } 3 \text{ times do begin for } 5 \text{ times do } a; S; C \text{ end}$$

$$\Rightarrow a; \text{for } 3 \text{ times do begin for } 5 \text{ times do } a; a; C \text{ end}$$

$$\Rightarrow a; \text{for } 3 \text{ times do begin for } 5 \text{ times do } a; a; S \text{ end}$$

$$\Rightarrow a; \text{for } 3 \text{ times do begin for } 5 \text{ times do } a; a; a \text{ end}$$