4. **Problem:** Pattern expressions are a generalisation of regular expressions used e.g. in some text editing tools of UN*X operating systems. In addition to the usual regular expression constructs, a pattern expression may contain string variables, including the constraint that any two appearances of the same variable must correspond to the same substring. Thus e.g. $abX^*Xa$ 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\}. Prove that pattern expressions are a proper generalisation of regular expressions, i.e. that pattern expressions can be used to describe also some nonregular languages.

**Answer:**

Consider the pattern expression $XX$. This expression denotes the language $L = \{zz \mid z \in \{a, b\}^*\}$. Suppose that $L$ is regular. Then, the pumping lemma for regular languages holds for it:

**Lemma:** If $L$ is a regular language, then there exists an integer $n > 0$ such that for each string $x \in n$ it holds that if $|x| \geq n$, then $x = uvw$ where (1) $|vw| \leq n$, (2) $|v| > 0$, and (3) $uv^k w \in L$ for every $k \in \mathbb{N}$.

Let us examine the string $x = a^nba^n b \in L$. As $|x| = 2n + 2 > 0$, there has to be a partition of $x$ into three parts such that all three conditions of the lemma are satisfied.

All partitions that satisfy (1) are of the form:

- $u = a^i$
- $v = a^j$
- $w = a^{n-i-j}ba^n b$

where $i + j \leq n$. From (2) we know that $j > 0$. Next we examine if we can find some values for $i$ and $j$ such that (3) also holds for $k = 0$:

$$uv^0 w = uvw = a^i a^{n-i-j}ba^n b = a^{p-j}ba^n b.$$

Since $j > 0$, $p - j < p$ so $uv^0 w \notin L$ for any choice of $i$ and $j$. Thus, $L$ is not regular.

Since we can define $L$ using pattern expressions, we now know that pattern expressions are strictly more expressive than regular expressions.

5. **Problem:** Prove that the language $L = \{w \mid w \text{ contains equally many } a's \text{ as } b's \}$ is not regular.

**Solution:**

**Lemma:** If $L$ is a regular language, then there exists an integer $n > 0$ such that for each string $x \in n$ it holds that if $|x| \geq n$, then $x = uvw$ where (1) $|uw| \leq n$, (2) $|v| > 0$, and (3) $uv^k w \in L$ for every $k \in \mathbb{N}$.

Consider $x = a^ib^n \in L$. If $L$ is regular, then we can divide $x$ into three parts $u, v,$ and $w$ such that all three conditions of the lemma hold. All partitions that satisfy (1) are of the form:

- $u = a^i$
- $v = a^j$
- $w = a^{n-(i+j)}ba^n$
where \( i + j \leq n \). From (2) we know that \( j > 0 \). Next we examine if we can find some values for \( i \) and \( j \) such that (3) also holds for \( k = 0 \):

\[
uw^0w = uw = a^i a^{n-(i+j)} b a^n b = a^{p-j} b^n \notin L.
\]

Since \( uw^0w \notin L \) for any \( i \) and \( j \), \( L \) is not regular.

6. **Problem:** Design an algorithm for testing whether a given a context-free grammar \( G = (V, \Sigma, P, S) \), generates a nonempty language, i.e. whether any terminal string \( x \in \Sigma^* \) can be derived from the start symbol \( S \).

**Solution:**

The following procedure \(?\text{GeneratesNonemptyLanguage}(G)\) takes a context-free grammar \( G \) as its input and it returns the value `true`, if the language \( L(G) \) is not empty.

\[
?\text{GeneratesNonemptyLanguage}(G = (V, \Sigma, P, S) : \text{context-free grammar})
\]

\[
T \leftarrow \Sigma
\]

repeat \( |V - \Sigma| \) times

for each \( A \rightarrow X_1 \cdots X_k \in P \)

if \( A \notin T \land X_1 \cdots X_k \in T^k \)

\[
T \leftarrow T \cup \{A\}
\]

if \( S \in T \)

return `true`

else

return `false`

The basic idea is to start from the set \( T = \Sigma \) of terminal symbols and then check whether it is possible to “retreat” to \( S \) using productions of \( P \) reversed. At each step a nonterminal \( A \) is added to the set \( T \) if there exists some rule for \( A \) such that all symbols in the right side belong to \( T \). These steps are repeated \( |V - \Sigma| \) times.

To see why \( |V - \Sigma| \) steps are enough, let us consider the word \( z \in L(G) \) such that \( z \) has the smallest parse tree of all words in \( L(G) \). If \( z \) has a derivation of the form:

\[
S \rightarrow^* uAy \rightarrow^* uvAxy \rightarrow^* uvwxy
\]

where \( u, v, w, x, y \in \Sigma^* \), then also \( z' = uwxy \) can be derived using the rules of the grammar\(^1\). In that case, the parse tree of \( z' \) is smaller than that of \( z \) contradicting our earlier assumption. Now we see that in the minimal parse tree of \( z \) it is not possible to have two occurrences of a nonterminal \( A \) in a single branch so we have to iterate over the set \( T \) only as many times as there are nonterminals in the grammar.

Consider the grammar \( G \):

\[
S \rightarrow BAB \mid ABA
\]

\[
A \rightarrow aAS \mid bBa
\]

\[
B \rightarrow bBS \mid c
\]

The computation of \( T \) proceeds as follows:

\[
T_0 = \{a, b, c\}
\]

\[
T_1 = \{a, b, c, B\}
\]

\[
T_2 = \{a, b, c, A, B\}
\]

\[
T_3 = \{a, b, c, A, B, C, S\}
\]

\[
(S \rightarrow BAB, S \rightarrow ABA)
\]

Since \( |V - \Sigma| = 3 \), the algorithm terminates and \( T = T_3 \) so \( L(G) \) is not empty. The smallest parse-tree of a \( z \in L(G) \) is:

\(^1\)Compare this with the pumping theorem of context-free languages.
Appendix: Chomsky normal form and CYK-algorithm

Let’s change the grammar:

\[ P = \{ S \rightarrow aAS \mid bBS \mid \varepsilon \} \]
\[ A \rightarrow aAA \mid b, \]
\[ B \rightarrow bBB \mid a \]

into Chomsky normal form, and check with CYK-algorithm whether words \textit{abb} and \textit{abba} belong to language \( L(G) \).

A grammar is in Chomsky normal form, if the following conditions are met:

1. Only the initial symbol \( S \) can generate an empty string.
2. The initial symbol \( S \) does not occur in the right hand side of any rule.
3. All rules are of form \( A \rightarrow BC \) or \( A \rightarrow a \) (where \( A, B \) ja \( C \) are nonterminals and \( a \) a terminal symbol), except for rule \( S \rightarrow \varepsilon \) (if such a rule exists).

The grammar is put into the normal form in phases.

1. **Initial symbol is removed from right side of the rules.**
   Because there are rules \( S \rightarrow aAS \) and \( S \rightarrow bBS \) in the grammar, let’s add a new starting symbol \( S' \) and a rule \( S' \rightarrow S \). The resulting set of rules is
   \[ S' \rightarrow S, \]
   \[ S \rightarrow aAS \mid bBS \mid \varepsilon \]
   \[ A \rightarrow aAA \mid b, \]
   \[ B \rightarrow bBB \mid a \]

2. **\( \varepsilon \)-productions are removed.**
   Because in the Chomsky normal form only the initial symbol \( S' \) may generate \( \varepsilon \), other \( \varepsilon \) rules must be removed from the grammar. We start by computing the set of erasable nonterminals: \( \text{NULL} \):
   \[ \text{NULL}_0 = \{ S \} \]
   \[ \text{NULL}_1 = \{ S, S' \} \]
   \[ \text{NULL}_2 = \{ S, S' \} = \text{NULL} \]

   Next, the rules \( A \rightarrow X_1 \cdots X_n \) are replaced by a set of rules
   \[ A \rightarrow \alpha_1 \cdots \alpha_2, \quad \text{where } \alpha_i = \begin{cases} X_i, & X_i \notin \text{NULL} \\ X_i \text{ or } \varepsilon, & X_i \in \text{NULL} \end{cases} \]

   Finally, we remove all rules of form \( A \rightarrow \varepsilon \) (except for rule \( S' \rightarrow \varepsilon \)). As the result we get rule set\(^2\):

\(^2\)To be exact, now we should add a new initial symbol \( S'' \) and rules \( S'' \rightarrow \varepsilon | S' \), but in this case we can use \( S' \) as the starting symbol without problems.
$S' \rightarrow S \mid \varepsilon$
$S \rightarrow aAS \mid aA \mid bBS \mid bB$
$A \rightarrow aAA \mid b,$
$B \rightarrow bBB \mid a$

3. **Unit productions are removed.**

Next we remove from the grammar all rules of form $A \rightarrow B$ where both $A$ and $B$ are nonterminals.

First, we compute sets $F(A)$ for all $A \in V - \Sigma$:

$F(A) = F(B) = F(S) = \emptyset$
$F(S') = \{S\}$

Nonterminal $B$ belongs to set $F(A)$ exactly when we can derive $B$ from $A$ using only unit productions:

Rule $A \rightarrow B$ is replaced by $\{A \rightarrow w \mid \exists C \in F(B) \cup \{B\} : C \rightarrow w \in P\}$. As the result we get a set of rules

$S' \rightarrow aAS \mid aA \mid bBS \mid bB \mid \varepsilon$
$S \rightarrow aAS \mid aA \mid bBS \mid bB$
$A \rightarrow aAA \mid b,$
$B \rightarrow bBB \mid a$

4. **Too long productions are removed.**

In the last phase we add into the grammar a new nonterminal $C_\sigma$ and a rule $C_\sigma \rightarrow \sigma$ for all $\sigma \in \Sigma$ and divide all rules $A \rightarrow w (|w| > 2)$ into a chain of rules, all of which consist of exactly two symbols.

The Chomsky normal form for the given grammar is the following set of rules:

$S' \rightarrow C_aS_1' \mid C_aA \mid C_bS_2' \mid C_bB \mid \varepsilon$
$S_1' \rightarrow AS$
$S_2' \rightarrow BS$
$S \rightarrow C_aS_1 \mid C_aA \mid C_bS_2 \mid C_bB$
$S_1 \rightarrow AS$
$S_2 \rightarrow BS$
$A \rightarrow C_aA_1 \mid b$
$A_1 \rightarrow AA$
$B \rightarrow C_bB_1 \mid a$
$B_1 \rightarrow BB$
$C_a \rightarrow a$
$C_b \rightarrow b$

Using CYK-algorithm we can check whether word $x = x_1 \cdots x_n$ belongs to the language defined by grammar $G$. During the progress of algorithm we compute nonterminal sets $N_{i,j}$. Set $N_{i,j}$ includes all those nonterminals, which can be used to derive substring $x_i \cdots x_j$. We can apply dynamic programming for computing the sets:

$N_{i,i} = \{A \mid (A \rightarrow x_i) \in P\}$
$N_{i,i+k} = \{A \mid \exists B, C \in V - \Sigma \text{ s. t. } (A \rightarrow BC) \in P \text{ and } \exists j : i \leq j < i + k \text{ s. e } B \in N_{i,j} \wedge C \in N_{j+1,i+k}\}$
Let’s look at the grammar we got above and word \textit{abba}. First we compute sets \( N_{i,i}, i \leq 4 \):

\[
\begin{array}{c|cccc}
 i \rightarrow & 1 : a & 2 : b & 3 : b & 4 : a \\
\hline
 k \downarrow & \text{abba} & \text{abba} & \text{abba} & \text{abba} \\
 0 & \{ B, C_a \} & \{ A, C_b \} & \{ B, C_a \} & \{ A, C_b \} \\
 1 & \{ \text{S}' \}, \text{S} & \{ A, S \} & \text{abba} & \text{abba} \\
\end{array}
\]

On each square of the array it has been denoted, which substring the square corresponds to.

Next we compute \( N_{1,2} \). Now the only possible \( j = 1 \), so we look at sets \( N_{1,1} = \{ B, C_a \} \) \( N_{2,2} = \{ A, C_b \} \). The only rules of form \( A \rightarrow BC \), \( B \in N_{1,1} \) and \( C \in N_{2,2} \), are: \( \{ S' \rightarrow C_aA, S \rightarrow C_aA \} \), so \( N_{1,2} = \{ S', S \} \). The same way we can compute sets \( N_{2,3} = \{ A_1 \} \) and \( N_{3,4} = \{ S', S \} \), so the second row of the array is

\[
\begin{array}{c|cccc}
 i \rightarrow & 1 : a & 2 : b & 3 : b & 4 : a \\
\hline
 k \downarrow & \text{abba} & \text{abba} & \text{abba} & \text{abba} \\
 0 & \{ B, C_a \} & \{ A, C_b \} & \{ B, C_a \} & \{ A, C_b \} \\
 1 & \{ \text{S}' \}, \text{S} & \{ A, S \} & \text{abba} & \text{abba} \\
\end{array}
\]

At square \( N_{1,3} \) we have to look at two alternatives,

\[
j = 1 \quad \Rightarrow \quad N_{1,1} = \{ C_a, B \} \quad j = 2 \quad \Rightarrow \quad N_{1,2} = \{ S', S \} \quad j = 2 \quad \Rightarrow \quad N_{1,3} = \{ C_b, A \}
\]

The nonterminal set corresponding to case \( j = 1 \) is \( \{ A \} \) (\( A \rightarrow C_aA_1 \)) and that of case \( j = 2 \) is \( \emptyset \), so \( N_{1,3} = \{ A \} \). We can continue the same way and and get the final table

\[
\begin{array}{c|cccc}
 i \rightarrow & 1 : a & 2 : b & 3 : b & 4 : a \\
\hline
 k \downarrow & \text{abba} & \text{abba} & \text{abba} & \text{abba} \\
 0 & \{ B, C_a \} & \{ A, C_b \} & \{ B, C_a \} & \{ A, C_b \} \\
 1 & \{ \text{S}' \}, \text{S} & \{ A, S \} & \text{abba} & \text{abba} \\
 2 & \{ \text{S}' \}, \text{S} & \{ A_1 \} & \{ S', S \} & \text{abba} \\
 3 & \{ \text{S}' \}, \text{S} & \{ A \} & \text{abba} & \text{S}' \}
\]

Since \( S' \in N_{1,4} \), \( \text{abba} \in L(G) \). But, \( S' \notin N_{1,3} \), so \( \text{abb} \notin L(G) \).