1.33 Let $L_1, L_2 \subseteq \{a, b\}^*$. a. Assume $L_1 \subseteq L_2$. Then $L_1^2 = L_1L_1 \subseteq L_1L_2 \subseteq L_2^2$ and in general $L_1^k = L_1^{k-1}L_1 \subseteq L_1^{k-1}L_2 \subseteq L_2^k$. Consequently, 
$L_1^* = \{\Lambda\} \cup L_1 \cup L_1^2 \cup \ldots \cup L_1^k \cup \ldots \subseteq \{\Lambda\} \cup L_2 \cup L_2^2 \cup \ldots \cup L_2^k \cup \ldots = L_2^*$. 
b. $L_1^* \cup L_2^* \subseteq (L_1 \cup L_2)^*$ always holds, since $L_1 \subseteq L_1 \cup L_2$ which implies that $L_1^* \subseteq (L_1 \cup L_2)^*$ (see item a. above) and similarly $L_2^* \subseteq (L_1 \cup L_2)^*$. 
c. The inclusion $L_1^* \cup L_2^* \subseteq (L_1 \cup L_2)^*$ may be strict: for $L_1 = \{0\}$ and $L_2 = \{1\}$ we have $\{0\}^* \cup \{1\}^* \neq \{0, 1\}^*$. 
d. If $L_1 \subseteq L_2^*$ then $L_1^* \cup L_2^* = L_2^*$ and since $L_1 \subseteq L_1 \cup L_2$ also $(L_1 \cup L_2)^* \subseteq (L_1 \cup L_2)^* = L_2^*$. Similarly, $L_2 \subseteq L_1^*$ implies that $L_1^* \cup L_2^* = L_1^* = (L_1 \cup L_2)^*$. 
Next consider $L_1 = \{0^2, 0^3\}$ and $L_2 = \{0^3, 0^5\}$. Then $L_1^* = \{\Lambda, 0^2, 0^4, 0^5, \ldots\} = \{0\}^* - \{0, 0^3\}$ and 
$L_2^* = \{\Lambda, 0^3, 0^5, 0^6, 0^8, 0^9, \ldots\} = \{0\}^* - \{0, 0^2, 0^4, 0^7\}$. Thus neither $L_1 \subseteq L_2^*$ nor $L_2 \subseteq L_1^*$. However, $L_1 \cup L_2^* = \{0\}^* - \{0\}$ and also $(L_1 \cup L_2)^* = \{0^2, 0^3, 0^5\}^* = \{\Lambda, 0^2, 0^3, 0^4, 0^5, \ldots\} = \{0\}^* - \{0\}$.

1.36 $L$ consists of all strings from $\{a, b\}^*$ that do not end with $b$ and do not have a subword $bb$. 
a. $L = \{a, ba\}^*$. 
b. Consider now the language $K$ consisting of all strings from $\{a, b\}^*$ that do not have a subword $bb$. Assume that $K = S^*$ for a finite set $S$. Then $b \in K = S^*$. Since $S^* = S^*S^* = S^* = K$, a contradiction. Hence there cannot exist a finite $S$ such that $K = S^*$.

1.37 Let $L_1, L_2, L_3 \subseteq \Sigma^*$ for some alphabet $\Sigma$. 
a. $L_1(L_2 \cap L_3) \subseteq L_1L_2 \cap L_1L_3$, because $w \in L_1(L_2 \cap L_3)$ implies that $w = xy$ with $x \in L_1$ and $y \in L_2 \cap L_3$. Consequently, $w \in L_1L_2$ and $w \in L_1L_3$. 

Answer to selected exercises from the textbook

John Martin, Introduction to Languages and the Theory of Computation

M.M. Bonsangue (and J. Kleijn)

December 3, 2014
Equality does not necessarily hold. Let $L_1 = \{a, ab\}$, $L_2 = \{ba\}$, and $L_3 = \{a\}$. Then $L_1(L_2 \cap L_3) = \emptyset \neq \{aba\} = \{aba, abba\} \cap \{aa, aba\} = L_1L_2 \cap L_1L_3$.

b. $L_1^* \cap L_2^* \supseteq (L_1 \cap L_2)^*$, because $w \in (L_1 \cap L_2)^*$ implies that $w$ is a concatenation of 0 or more words from $L_1 \cap L_2$. Consequently, $w \in L_1^*$ and $w \in L_2^*$.

Equality does not necessarily hold. Let $L_1 = \{a\}$ and $L_2 = \{aa\}$. Then $L_1^* \cap L_2^* = \{a\}^* \cap \{aa\}^* = \{aa\}^* \neq \{a\}^* = (L_1 \cap L_2)^*$.

c. $L_1L_2^*$ and $(L_1L_2)^*$ are not necessarily included in one another. Let $L_1 = \{a\}$ and $L_2 = \{b\}$. Then $L_1^*L_2^* = \{a\}^*\{b\}^*$ consisting of words with a number of $a$’s followed by some number of $b$’s and $(L_1L_2)^* = \{ab\}^*$ consisting of words with alternating $a$’s and $b$’s. These two languages are incomparable: $aab \in L_1^*L_2^* - (L_1L_2)^*$ and $abab \in (L_1L_2)^* - L_1^*L_2^*$. 


2.2

(a) All strings containing a substring starting with at least two a’s followed by \(ba\).
(b) All strings ending with at least two a’s followed by \(ba\).
(c) All strings starting with \(aaba\).
(d) Either the empty string or all strings starting with \(a\) and ending with \(b\).
(d) All strings with no two consecutive equal alphabet symbols (thus no \(aa\) or \(bb\)).

2.10 a., b. and c. The desired automata each have \((A, X)\) as initial state.

Next we apply the product construction to \(M_1\) and \(M_2\):

\[
\begin{array}{c|cc}
   & a & b \\
\hline
(A, X) & (B, X) & (A, Y) \\
(B, X) & (B, X) & (C, Y) \\
(A, Y) & (B, X) & (A, Z) \\
(C, Y) & (B, X) & (A, Z) \\
(A, Z) & (B, Z) & (A, Z) \\
(B, Z) & (B, Z) & (C, Z) \\
(C, Z) & (B, Z) & (A, Z) \\
\end{array}
\]

Note that the states \((B, Y)\) and \((C, X)\) which are not reachable from \((A, X)\) are not mentioned in the table for the product transition function (see exercise 3.29).

For \(L_1 \cup L_2\), we have as accepting states \{\((C, Y)\), \((C, Z)\), \((A, Z)\), \((B, Z)\)\}, that is any pair of original states in which at least one is accepting.

For \(L_1 \cap L_2\), we have as accepting states \{\((C, Z)\)\}, that is any pair of original states in which both are accepting.

For \(L_1 - L_2\), we have as accepting states \{\((C, Y)\)\}, that is any pair of original states in which the first is accepting and the second not.

Drawing the automata is now easy.

2.11 Let \(M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)\) be an FA and let \(M_2 = (Q_2, \Sigma, q_2, A_2, \delta_2)\).

Define for all \(p \in Q_1\), \(q \in Q_2\), and \(a \in \Sigma\): \(\delta((p, q), a) = (\delta_1(p, a), \delta_2(q, a))\).

We have to prove that \(\delta^*(p, q, x) = (\delta_1^*(p, x), \delta_2^*(q, x))\) for all \(p \in Q_1\), \(q \in Q_2\), and \(x \in \Sigma^*\) (see the proof of Theorem 2.15).

We use induction on \(|x|\), the length of \(x\).

If \(|x| = 0\), then \(x = \Lambda\) and we have \(\delta^*((p, q), \Lambda) = (p, q) = (\delta_1^*(p, \Lambda), \delta_2^*(q, \Lambda))\) by the inductive definitions of \(\delta^*\), \(\delta_1^*\), and \(\delta_2^*\).

Let \(|x| = n + 1\). Then \(x = ya\) for some \(y \in \Sigma^*\) and \(a \in \Sigma\). Hence \(|y| = n\) and according to the induction hypothesis: \(\delta^*((p, q), y) = (\delta_1^*(p, y), \delta_2^*(q, y))\).
Consequently, \( \delta^*((p,q),x) = \delta^*((p,q),ya) = \delta(\delta^*((p,q),y)a) \) by the inductive definition of \( \delta^* \). By the induction hypothesis \( \delta(\delta^*((p,q),y),a) = \delta((\delta^*_1(p,y),\delta^*_2(q,y)),a) \) and \( \delta(\delta^*_1(p,y),\delta^*_2(q,y)),a) = (\delta_1((\delta^*_1(p,y),\delta^*_2(q,y)),a)) \) by the definition of \( \delta \). Finally, using the inductive definition of \( \delta^*_1 \) and \( \delta^*_2 \), we derive \( (\delta_1((\delta^*_1(p,y),a),\delta_2((\delta^*_2(q,y),a))) = (\delta^*_1(p,ya),\delta^*_2(q,ya)) = (\delta^*_1(p,x),\delta^*_2(q,x)) \) as desired.

2.13 Consider the FA from Figure 2.17d. It accepts the language \( L \) consisting of all strings from \( \{a,b\}^\ast \) that do not contain \( aa \) and do not end in \( ab \). The simplest strings corresponding to its four states are \( \Lambda, a, ab, \) and \( aa \). If any two of these strings are distinguishable, then it follows from Theorem 2.21 that any FA recognizing \( L \) has at least 4 states.

for \( \Lambda \) and \( a \) and for \( ab \) and \( aa \), choose \( z = a \):
\( \Lambda a \in L, aba \in L \), but \( aa \notin L \) and \( aaa \notin L \);
for \( \Lambda \) and \( ab \), for \( \Lambda \) and \( aa \), for \( a \) and \( ab \), and for \( a \) and \( aa \), choose \( z = \Lambda \):
\( \Lambda a \in L, a\Lambda \in L \), but \( ab\Lambda \notin L \) and \( aa\Lambda \notin L \).
Hence, there is no FA accepting \( L \) with fewer states than the given FA.

2.14 Let \( z \) be a word over the alphabet \( \{a,b\} \). Any FA which accepts \( L = \{a,b\}^\ast \{z\} \) has at least \( |z| + 1 \) states. The reason is that \( z \) has \( |z| + 1 \) prefixes (from \( \Lambda \) to \( z \) itself) which all have to be distinguished in the automaton. If \( z = xy \) and \( z = uv \) with \( |x| > |u| \), then \( xy \in L \), but \( uy \notin L \). Hence \( x \) and \( u \) are distinguishable with respect to \( L \).
No more states are needed, since there is no need to distinguish between a word and the longest prefix of \( z \) which is a suffix of this word:
Consider a word \( x \) and let \( v \) be its longest suffix such that \( x = uv \) where \( v \) is such that \( z = vw \) for some \( w \). (Note that every word \( x \) has such a suffix!)
Then for all words \( y \in \{0,1\}^\ast \) we have
case \( |y| \geq |z| \): \( xy \in L \) if and only if \( vy \in L \) because it is only relevant whether \( y \) ends with \( z \) or not;
or case \( |y| < |z| \): then \( xy \in L \) if and only if \( uvy \in L \) if and only if \( vy \) ends with \( z \) if and only if \( vy \in L \).
Consequently, \( x \) and \( v \) are indistinguishable and we know how to define an FA accepting \( L \).
As an example we give an FA accepting \( \{a,b\}^\ast \{babb\} \) with 5 states:
\( q_0 \) corresponding to matching suffix \( \Lambda \); \( q_1 \) for \( b \); \( q_2 \) for \( ba \); \( q_3 \) for \( bab \); and \( q_4 \) for \( babb \), this is the final state since its last symbols form \( z = babb \).
2.33 $L = \{x\}$ with $x$ a word over $\{a, b\}$. Then $I_L$ has $|x| + 2$ equivalence classes: one for each prefix of $x$ and one containing all the words that are not a prefix of $x$.

2.36 $L \subseteq \{a, b\}^*$ is a language for which we are asked to give a finite automaton. We apply the construction used to prove Theorem 5.1. $I_L$ has 4 equivalence classes $[\Lambda]$, $[a]$, $[ab]$, and $[b]$ which we will use as states. $[\Lambda]$ will be the initial state. Moreover, since $ab \in L$ and $\Lambda, a, b \not\in L$, we designate $[ab]$ as the only final state of the automaton. The transition function $\delta$ is defined as follows.

\[ \delta([\Lambda], a) = [a] \quad \text{and} \quad \delta([\Lambda], b) = [b]; \]
\[ \delta([a], b) = [ab] \quad \text{and} \quad \text{since} \ a \ I_L \ aa, \ \text{we set} \ \delta([a], a) = [aa] = [a]; \]
\[ \text{similarly,} \ \delta([ab], a) = [aba] = [b] \quad \text{and} \quad \delta([ab], b) = [abb] = [a]. \]

What remains are the transitions from $[b]$. We know that $b$ is not a prefix of any word in $L$. Hence (again using exercise 5.4), $[b]$ is the equivalence class consisting of all words which can never be extended to a word in $L$. In the automaton this equivalence class is a sink: $\delta([b], a) = [ba] = [b]$ and $\delta([b], b) = [bb] = [b]$.

(Draw the automaton.)

2.55 See figure 2.45.

a. We construct $S = \{(p, q) \subseteq Q \times Q \mid p \neq q\}$ recursively, following Algorithm 2.40. In the first pass we note that 5 is an accepting state and the other states are not. Thus we mark (with 1) in the $Q \times Q$ table, the entries $(5, 1)$, $(5, 2)$, $(5, 3)$, and $(5, 4)$. (For symmetry reasons it is sufficient to fill in only the lower triangle. At the diagonal, we have the identities $(p, p)$ which are never in $S$).

In the second pass (given as 2), we find:

- $(1, 2) \in S$, because $\delta(1, b) = 3$ and $\delta(2, b) = 5$, and $(3, 5) \in S$;
- $(2, 3) \in S$, because $\delta(2, b) = 5$ and $\delta(3, b) = 3$, and $(5, 3) \in S$;
- $(1, 4) \in S$, because $\delta(1, b) = 3$ and $\delta(4, b) = 5$, and $(3, 5) \in S$;
- $(3, 4) \in S$, because $\delta(3, b) = 3$ and $\delta(4, b) = 5$, and $(3, 5) \in S$;

and no more.

In the third pass, we find no new pairs.
Consequently, we have the following equivalence classes: \{1, 3\}, \{2, 4\} and \{5\}, which gives a minimal FA with three states \(q_0 = \{1, 3\}\), the initial state; \(q_1 = \{2, 4\}\); and \(q_3 = \{5\}\), the only final state. Its transition function is given in the next table:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{1, 3}</td>
<td>(q_0)</td>
</tr>
<tr>
<td>1</td>
<td>(q_1)</td>
<td>(q_0)</td>
</tr>
<tr>
<td>2</td>
<td>(q_0)</td>
<td>(q_3)</td>
</tr>
<tr>
<td>3</td>
<td>(q_1)</td>
<td>(q_3)</td>
</tr>
</tbody>
</table>

Draft this FA.

b. The given FA is already minimal.

2.57 You are asked for a number of languages over the alphabet \(\{a, b\}\) to determine whether they can be accepted by an FA or not. If you do well (go after the number of equivalence classes, use the pumping lemma or try to make up an FA for the language) then only given the language in b. is regular.

a. Let \(L = \{x \in \{a, b\}^* \mid \exists w, y \in \{a, b\}^*: w \neq \Lambda \land x = wwy\}\), and take two words \(v = ab^n\) and \(v' = ab^m\) for \(n, m > 0\) and \(n \neq m\). For \(z = ab^n\) we have that \(vz = ab^nab^n \in L\) whereas \(v'z = ab^mab^n \notin L\). It follows that all words \(ab^n\) for \(n > 0\) are pairwise distinguishable, and therefore there can be no FA recognizing \(L\), as it would need infinitely many states by Theorem 2.26.

b. Let \(L = \{x \in \{a, b\}^* \mid \exists y, z: w \neq \Lambda \land x = ywz\}\).

Claim: \(x \in L\) if and only if \(x\) contains \(aa\), \(bb\), \(abab\) or \(baba\) as a subword.

Proof of the claim: if \(aa, bb, abab\) or \(baba\) a subword is of \(x\), dan is \(x\) by definition in \(L\).

Conversely, assume that \(x \in L\). Then \(|x| \geq 2\). Let \(aa\) and \(bb\) be not subwords of \(x\). Then in \(x\) it holds that every \(a\) that is not the last symbol of \(x\), is followed by a \(b\), en that every \(b\) that is not the last symbol of \(x\), is followed by an \(a\). Let now \(y, w, z \in \{a, b\}^*\) with \(w \neq \Lambda\) such that \(x = ywz\). Then we
know that \( w \neq 0 \) and \( w \neq 1 \); thus either \( w = abu \) or \( w = bau \). If \( u = \Lambda \), then we are done. Otherwise \( |u| \geq 2 \), and \( w = abav \) or \( w = babav \), respectively, and we are ready also in this case.

The only case that remains to check is when \( |u| = 1 \). If \( w = abu \), then must be \( u = a \) and \( x = ywwz = yabaabz \), contradicting our assumption that \( x \) does not contain \( aa \) as subword. Analogously, if \( w = bau \), then must be \( u = b \) and is \( x = ywwz = ybabbaz \), contradicting our assumption that \( x \) does not contain \( bb \) as subword.

Summarizing, if \( x \in L \), then \( x \) contains at least one of the words \( aa \), \( bb \), \( abab \) and \( baba \) as subword.

Now it is easy to draw a FA that recognizes \( L \):

![Diagram](attachment:image.png)
3.1 a. \( r = b^*(ab)^*a^* \): the word \( aab \) is not in the language, defined by \( r \), since every \( a \) should be followed by a \( b \) or belong to a suffix of \( a \)'s. Note that \( \Lambda, a, b, \) and all words of length 2 are in the language, defined by \( r \). So, \( aab \) is of minimal length.

Another example is \( abb \): every \( b \) should be preceded by a \( a \) unless it is part of a prefix of \( b \)'s.

d. \( r = b^*(a + ba)^*b^* \): the word \( abba \) does not belong to the language of \( r \), because that requires that a \( b \) can only be followed by a \( b \) if it belongs to a prefix or suffix consisting of \( b \)'s. Verify that all words of length \( \leq 3 \) belong to the language.

3.3 a. \( r(r^*r + r^*) + r^* = r^* \).
   b. \( (r + \Lambda)^* = r^* \).
   c. The expression \( (r + s)^*rs(r + s)^* + s^*r^* \) denotes all words that contain at least once \( rs \) (i.e. the expression \( (r + s)^*rs(r + s)^* \)) or do not contain any occurrence of \( rs \) at all (i.e. the expression \( s^*r^* \)). This is thus equivalent to \( (r + s)^* \).

3.6 a. \((w)^*(z)^*\).
   b. \((w)^*a(w + z)^*\).
   c. \((w + z)^*(a + \Lambda)\).

3.7 a. \( b^*ab^*ab^* \).
   b. \( b^*ab^*a(a + b)^* \).
   c. \( (a + b)^*(a + aa + ba + bb) + \Lambda \).
   f. \( (b^*ab^*ab^*)^* \).
   k. \( (a + ba)^*b^* \).
   l. \( (a + b)^*bab(a + b)^*aba(a + b)^* + (a + b)^*aba(a + b)^*bab(a + b)^* + (a + b)^*bab(a + b)^* \).
   m. \( (aa + bb + abab + baba + abba + baab)^*b \).
   n. \( (aa + bb + abab + baba + abba + baab)^*(ab + ba) \).

3.10 The reverse function \( \text{rev} \) assigns to each string its reversal (mirror image).

Formally, given an alphabet \( \Sigma \), we define \( \text{rev} : \Sigma^* \rightarrow \Sigma^* \) recursively by:

\( \text{rev}(\Lambda) = \Lambda \) (no change)

\( \text{rev}(xa) = a\text{rev}(x) \) for \( x \in \Sigma^* \), \( a \in \Sigma \) (last letter first, reverse the rest)

\( \text{rev}(x) \) may be abbreviated as \( x^r \).

For a language \( L \) we use \( L^r \) to denote the language consisting of the reversals of the words from \( L \), thus \( L^r = \{ x^r \mid x \in L \} \).
a. Consider the regular expression \( e = (aab + bbaba)^*baba \) defining the regular language \( |e| \). Then the language \( |e|^r \) can be defined by the regular expression \( e^r = abab(baa + ababb)^* \); thus \( |e|^r = |e|^r \).

b. In general we have the recursively defined function \( rrev \) which “reverses” regular expressions (in the sense that it yields a regular expression with a reversed semantics): \( rrev(\emptyset) = \emptyset \); \( rrev(\Lambda) = \Lambda \); \( rrev(a) = a \) for all \( a \in \Sigma \).

Now we have to prove that this \( rrev \) has the property \( |rrev(e)| = |e|^r \).

This is proved by induction on the structure of \( e \):

- \( e = \emptyset \): then \( |rrev(\emptyset)| = ||\emptyset|| = ||\emptyset||^r \);
- \( e = \Lambda \): then \( |rrev(\Lambda)| = ||\Lambda|| = ||\Lambda||^r \);
- \( e = a \): then \( |rrev(a)| = ||a|| = ||a||^r \).

Induction step, assuming that \( |rrev(e_1)| = |e_1|^r \) and \( |rrev(e_2)| = |e_2|^r \):

- \( e = e_1 + e_2 \): then
  \[ |rrev(e_1 + e_2)| = |rrev(e_1)| + |rrev(e_2)| = |rrev(e_1)| + |rrev(e_2)| = (\text{induction}) \]
  \[ (||e_1||^r + ||e_2||^r) = (|e_1|^r + |e_2|^r) = ||e_1 + e_2||^r \];

- \( e = e_1 e_2 \): then
  \[ |rrev(e_1 e_2)| = |rrev(e_2) rrev(e_1)| = |rrev(e_2)| \cdot |rrev(e_1)| = (\text{induction}) \]
  \[ (||e_2||^r)(||e_1||^r) = (||e_1||^r) \cdot (||e_2||^r) = ||e_1 e_2||^r \];

- \( e = e_1^* \): then
  \[ |rrev(e_1^*)| = ||rrev(e_1)^*|| = ||rrev(e_1)||^* = (\text{induction}) \]
  \[ (||e_1||^r)^* = (||e_1||^r)^* = ||e_1^*||^r \].

c. It follows from b. that the language \( L^r \) is regular whenever the language \( L \) is regular: we have seen that \( L = |e| \) implies that \( L^r = |rrev(e)| \) and that \( rrev(e) \) is a regular expression follows immediately from the definition of \( rrev \) as given above.

3.18 See Figure 3.34.

a. We determine \( \delta^*(1, aba) \). First observe that \( \delta^*(1, \Lambda) = \Lambda(\{1\}) = \{1\} \); then \( \delta^*(1, a) = \Lambda(\bigcup_{r \in \delta^*(1, \Lambda)} \delta(r, a) = \Lambda(\delta(1, a)) = \Lambda(\{2\}) = \{2, 3\} \). This means that processing symbol \( a \) from the initial state leads to state 2 or state 3.

We add \( b \): \( \delta^*(1, ab) = \Lambda(\delta(2, b) \cup \delta(3, b)) = \Lambda(\emptyset \cup \{3, 4\}) = \{3, 4, 5\} \) and so after \( ab \) we are in either state 3 or state 4 or state 5.

Finally we process another \( a \): \( \delta^*(1, aba) = \Lambda(\delta(3, a) \cup \delta(4, a) \cup \delta(5, a)) = \Lambda(\{4\} \cup \{4\} \cup \emptyset) = \Lambda(\{4\}) = \{4, 5\} \). Thus after reading \( aba \) we are in state 4 or in state 5 and since 5 is an accepting state, \( aba \) is accepted by \( M \).

b. \( abab \) is not accepted: from a. we know that \( \delta^*(1, aba) = \{4, 5\} \). Thus
\(\delta^*(1, abab) = \Lambda(\delta(4, b) \cup \delta(5, b)) = \Lambda(\emptyset) = \emptyset\). Not only is there no accepting state for \(abab\), it cannot even be completely processed!

**c. \text{aaabbb}** is accepted by \(M\) (check!).

### 3.22

a. \(\Lambda\{2, 3\} = \{2, 3, 5\}\).

b. \(\Lambda\{1\} = \{1, 2, 5\}\).

d. To determine \(\delta^*(1, ba) = \) first observe that \(\delta^*(1, \Lambda) = \Lambda\{1\} = \{1, 2, 5\}\) (see above item b.). We thus have \(\delta^*(1, b) = \Lambda(\bigcup_{p \in \delta^*(1, \Lambda)} \delta(p, b)) = \Lambda(\delta(1, b) \cup \delta(2, b) \cup \delta(5, b)) = \Lambda\{6, 7\} = \{1, 2, 5, 6, 7\}\). Finally we obtain \(\delta^*(1, ba) = \Lambda(\bigcup_{p \in \delta^*(1, b)} \delta(p, a)) = \Lambda(\delta(1, a) \cup \delta(2, a) \cup \delta(5, a) \cup \delta(6, a) \cup \delta(7, a)) = \Lambda\{3, 5\} = \{3, 5\}\).

### 3.41 a.

We use the partial derivatives method to compute the transitions and the states of the *nondeterministic automaton* corresponding to the regular expression \(E_0 = (b + bba)^*a\). First we note that \(\Lambda \not\in L(E_0)\), so \(E_0\) is not an accepting state. We have \(\partial_a((b + bba)^*a) = \partial_a((b + bba)\Lambda a \cup \partial_a(a) = \partial_a((b + bba))(b + bba)^*a \cup \{\Lambda\} = (\partial_a(b) \cup \partial_a(bba))(b + bba)^*a \cup \{\Lambda\} = \{\Lambda\}\). This is a new state, clearly accepting, that we denote by \(E_1\). Continuing our calculation we obtain \(\partial_b((b + bba)^*a) = \partial_b((b + bba)\Lambda a \cup \partial_b(a) = \partial_b((b + bba))(b + bba)^*a \cup \emptyset = (\partial_b(b) \cup \partial_b(bba))(b + bba)^*a = \{\Lambda\} \cup \{a\} \cup \{b + bba\} \cup \{b + bba\}^*a = \{b + bba\}^*a, ba(b + bba)^*a\}. The first element in the set is \(E_0\), and we denote the other one by \(E_2\). Also here \(\Lambda \not\in L(E_2)\).

The transitions from \(E_1 = \Lambda\) are calculated as follows: \(\partial_a(\Lambda) = \partial_b(\Lambda) = \emptyset\).

The transitions from \(E_2 = ba(b + bba)^*a\) are: \(\partial_a(ba(b + bba)^*a) = \partial_a(b)(a + bba)^*a = \emptyset\) and \(\partial_b(ba(b + bba)^*a) = \partial_b(b)(a + bba)^*a = \{\Lambda\} \cup \{a\} \cup \{b + bba\}^*a = \{a + b + bba\}^*a\). The element of this set is a new state, say \(E_3\), with \(\Lambda \not\in (E_3)\), thus non accepting.

The transitions from \(E_3 = a(b + bba)^*a\) are: \(\partial_a(a(b + bba)^*a) = \partial_a(a)(b + bba)^*a = \{\Lambda\} \cup \{b + bba\}^*a\). (this is just \(E_0\) and \(\partial_b(a(b + bba)^*a) = \partial_b(a)(b + bba)^*a = \emptyset\)\).

The resulting automaton (with \(E_0\) as initial state) is summarized in the following table:

<table>
<thead>
<tr>
<th>(q)</th>
<th>(\delta(q, a))</th>
<th>(\delta(q, b))</th>
<th>Accepting?</th>
</tr>
</thead>
<tbody>
<tr>
<td>(E_0)</td>
<td>{(E_1)}</td>
<td>{(E_0, E_2)}</td>
<td>No</td>
</tr>
<tr>
<td>(E_1)</td>
<td>\emptyset</td>
<td>\emptyset</td>
<td>Yes</td>
</tr>
<tr>
<td>(E_2)</td>
<td>\emptyset</td>
<td>{(E_3)}</td>
<td>No</td>
</tr>
<tr>
<td>(E_3)</td>
<td>{(E_0)}</td>
<td>\emptyset</td>
<td>No</td>
</tr>
</tbody>
</table>
3.41 d. Let $E_0 = (a^*bb)^* + bb^*a^*$. We use the method of derivatives to find a deterministic finite automaton accepting the language $L(E_0)$. Since $\Lambda \in L(E_0)$, the state $E_0$ is accepting. Further we calculate the two derivatives: $D_a((a^*bb)^* + bb^*a^*) = D_a((a^*bb)^*) + D_a(bb^*a^*) = D_a(a^*bb)(a^*bb)^* + D_a(bb)bb^*a^* = (D_a(a^*bb) + D_a(bb))(a^*bb)^* + b^*a^* = (D_a(a^*bb) + D_a(bb))(a^*bb)^* + b^*a^* = (D_a(a^*bb) + D_a(bb))(a^*bb)^* + b^*a^* = (D_a(a^*bb) + D_a(bb))(a^*bb)^* + b^*a^* = (\emptyset + b)((a^*bb)^* + b^*a^*) = b(a^*bb)^* + b^*a^* = b(a^*bb) + b^*a^* = E_2$. Note that $\Lambda$ is in the language of $E_2$ but not in the languages of $E_1$. Thus only $E_2$ is accepting.

Next we calculate the derivatives of $E_1$. $D_a(a^*bb(a^*bb)^*) = D_a(a^*bb)(a^*bb)^* + D_a(bb(a^*bb)^*) = a^*bb(a^*bb)^* + \emptyset = a^*bb(a^*bb)^* = E_1$ and $D_b(a^*bb(a^*bb)^*) = D_b(a^*bb)(a^*bb)^* + D_b(bb(a^*bb)^*) = \emptyset + b(a^*bb)^* = b(a^*bb)^* = E_3$. Also $E_3$ is not accepting.

Next we calculate the derivatives of $E_2$. $D_a(b(a^*bb)^* + b^*a^*) = D_a(b(a^*bb)^*) + D_a(b^*a^*) = D_a(b)(a^*bb)^* + D_a(b^*a^*) = D_a(b)(a^*bb)^* + D_a(b)(b^*a^*) = D_a(b)(a^*bb)^* + D_a(b)(b^*a^*) = (a^*bb)^* + b^*a^* + \emptyset = (a^*bb)^* + b^*a^* = E_5$ (again, an accepting state!).

The derivatives of $E_3$ are: $D_a(b(a^*bb)^*) = D_a(b)(a^*bb)^* = \emptyset = E_6$ and $D_b(b(a^*bb)^*) = D_b(b)(a^*bb)^* = (a^*bb)^* = E_7$ (an accepting state).

The derivatives of $E_4$ are $D_a((a^*bb)^* + b^*a^*) = D_a((a^*bb)^*) + D_a(b^*a^*) = D_a(a^*bb)(a^*bb)^* + D_a(b^*a^*) = D_a(a^*bb)(a^*bb)^* + D_a(b^*a^*) = D_a(a^*bb)(a^*bb)^* + D_a(a^*bb)^* + a^* = a^*bb(a^*bb)^* + \emptyset + a^* = a^*bb(a^*bb)^* + a^* = E_8$ (an accepting state) and $D_b((a^*bb)^* + b^*a^*) = D_b((a^*bb)^*) + D_b(b^*a^*) = D_b(a^*bb)(a^*bb)^* + D_b(b^*a^*) = D_b(a^*bb)(a^*bb)^* + D_b(b^*a^*) = D_b(a^*bb)(a^*bb)^* + b^*a^* + \emptyset = b(a^*bb)^* + b^*a^* = E_2$.

We skip the calculation of the derivatives of the other states, which are either easy or can be derived by the above calculations. The resulting automaton (with $E_0$ as initial state) is summarized in the following table:
We add a new initial state $q_i \not\in Q$ and a $\Lambda$-transition $(q_i, \Lambda) = \{q_0\}$.
All the rest remains unchanged.

b. We add a new single accepting state $q_f \not\in Q$ and a $\Lambda$-transition $(q, \Lambda) = \{q_f\}$ from every $q \in A$. All the rest remains unchanged.

3.51a See Figure 3.40 (a). We use the algebraic method of Brzozowski to derive for the depicted automata a corresponding regular expression.
First we write the automaton in Figure 3.40 (a) as a system of 3 equations in three variables:

$$x_1 = ax_3 + bx_2$$
$$x_2 = ax_1 + bx_3$$
$$x_3 = ax_2 + bx_1 + \Lambda$$

By substituting $x_3$ in the first two equations we obtain the system

$$x_1 = a(ax_2 + bx_1 + \Lambda) + bx_2 = abx_1 + (aa + b)x_2 + a$$
$$x_2 = ax_1 + b(ax_2 + bx_1 + \Lambda) = bax_2 + ((a + bb)x_1 + b)$$

Using the Arden’s lemma, we obtain that $x_2 = (ba)^*((a + bb)x_1 + b)$. If we substitute $x_2$ in the first equations we have

$$x_1 = abx_1 + (aa + b)(ba)^*((a + bb)x_1 + b) + a$$

Using again Arden’s lemma, we obtain that $x_1 = ((ab + (aa + b)(ba)^*(a + bb))((aa + b)(ba)^*b + a)$.
This is a regular expression denoting the same language of the automaton in Figure 3.40 (a).

3.51b See Figure 3.40 (c). We use the state removal method of Brzozowski and McCluskey to derive for the depicted automata a corresponding regular

---

<table>
<thead>
<tr>
<th>$q$</th>
<th>RegExp</th>
<th>$\delta(q, a)$</th>
<th>$\delta(q, b)$</th>
<th>Accepting?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_0$</td>
<td>$(a^<em>bb)^</em> + bb^<em>a^</em>$</td>
<td>$E_1$</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>$E_1$</td>
<td>$(a^*bb)(a^<em>bb)^</em>$</td>
<td>$E_1$</td>
<td>$E_3$</td>
<td>No</td>
</tr>
<tr>
<td>$E_2$</td>
<td>$b(a^<em>bb)^</em> + b^<em>a^</em>$</td>
<td>$E_4$</td>
<td>$E_5$</td>
<td>Yes</td>
</tr>
<tr>
<td>$E_3$</td>
<td>$b(a^<em>bb)^</em>$</td>
<td>$E_6$</td>
<td>$E_7$</td>
<td>No</td>
</tr>
<tr>
<td>$E_4$</td>
<td>$a^*$</td>
<td>$E_4$</td>
<td>$E_6$</td>
<td>Yes</td>
</tr>
<tr>
<td>$E_5$</td>
<td>$(a^<em>bb)^</em> + b^<em>a^</em>$</td>
<td>$E_8$</td>
<td>$E_3$</td>
<td>Yes</td>
</tr>
<tr>
<td>$E_6$</td>
<td>$\emptyset$</td>
<td>$E_6$</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>$E_7$</td>
<td>$(a^<em>bb)^</em>$</td>
<td>$E_1$</td>
<td>$E_3$</td>
<td>Yes</td>
</tr>
<tr>
<td>$E_8$</td>
<td>$(a^<em>bb)(a^<em>bb)^</em> + a^</em>$</td>
<td>$E_8$</td>
<td>$E_3$</td>
<td>Yes</td>
</tr>
</tbody>
</table>
expression.
First we add a new initial state \( q_0 \) without incoming transitions and a new
(the only) final state \( q_f \) without outgoing transitions in such a way that the
resulting NFA accepts the same language (see exercise 3.44). At the same
time we combine with \(+\) the labels of parallel edges into a single regular
expression.

\[
\begin{align*}
1 & \rightarrow a, b \rightarrow 2 \\
& \downarrow a \quad b \\
4 & \leftarrow a, b \leftarrow 3 \\
\quad & \Lambda
\end{align*}
\]

\[
\begin{align*}
1 & \rightarrow a + b \rightarrow 2 \\
& \downarrow a \quad b \\
4 & \leftarrow a + b \leftarrow 3 \\
\quad & \Lambda
\end{align*}
\]

Now we remove state 4. Before deleting 4, we consider the transitions from
3 to 4 and from 4 to 1 and 2. This leads to the introduction of an arc labeled
with \((a + b)a\) from 3 to 1 and an arc labeled with \((a + b)b\) from 3 to 2.

\[
\begin{align*}
1 & \rightarrow a + b \rightarrow 2 \\
& \downarrow (a + b)a \quad a + b \\
3 & \leftarrow (a + b)b \leftarrow (a + b)a
\end{align*}
\]

Then state 1 is removed. This leads to the introduction of an arc labeled
with \(\Lambda(a + b)\) from \(q_0\) to 2 and an arc labeled with \((a + b)a(a + b)\) from 3
to 2. The latter is combined using \(+\) with the label \((a + b)b\) of the already
existing arc from 3 to 2.

\[
\begin{align*}
1 & \rightarrow a + b \rightarrow 2 \\
& \downarrow (a + b)a \quad a + b \\
3 & \leftarrow (a + b)b \leftarrow (a + b)a
\end{align*}
\]

Then state 3 is removed. This leads to the introduction of an arc labeled
with \((a + b)\Lambda\) from 2 to \(q_f\) which is combined with the existing parallel arc
labeled with \(\Lambda\). Also an arc from 2 to 2 is added which is labeled with
\((a + b)((a + b)b + (a + b)a(a + b))\), a combination of the label of the arc from
2 to 3 and that of the arc from 3 to 2.
Finally, we remove state 2 and find a regular expression for $L(M)$.

$$L(M) = \{a, b\}(\{a, b\}\{\{a, b\}\} \cup \{a, b\}\{a\}\{a, b\})^*\{\Lambda, a, b\}.$$
4.1 In each case say which language is generated by the CFG $G$ with the productions as indicated.

a. $L(G) = a, b^*.$

b. $L(G) = \{a, b\}^* \{a\}.$

c. $L(G) = \{ba\}^* \{b\}.$

d. $L(G) = \{x \in \{a, b\}^* | \text{bb does not occur in } x\}.$

e. $L(G) = \{a, b\}^* \{b\}.$

f. $L(G) = \{x \in \{a, b\}^* | x \text{ is a palindrome over } \{a, b\}\}.$

4.26 Describe the language generated by the given grammars.

a. $S \rightarrow aA | bC | b, \quad A \rightarrow aS | bB, \quad B \rightarrow aC | bA | a, \quad C \rightarrow aB | bS$

This is a regular grammar. Using the construction given in the proof of Theorem 4.14, we obtain the following NFA accepting $L(G)$.

Now it is not difficult to see that

$L(G) = \{x \in \{a, b\}^* | n_a(x) \text{ is even and } n_b(x) \text{ is odd}\}.$

4.3 Find a context-free grammar generating the given language.

a. For $L = \{xay \mid x, y \in \{a, b\}^* \land |x| = |y|\}$ the CFG with productions

$$S \rightarrow aSa | aSb | bSa | bSb | a,$$

b. For $L = \{xaay, xbyx^r \mid x, y \in \{a, b\}^* \land y = y^r\}$, i.e. the language of all words which are palindromes over $\{a, b\}$ with exactly one single “mistake”.

c. $L(G) = \{x \in \{a, b\}^* | |x| \text{ is even }\}.$

d. $L(G) = \{x \in \{a, b\}^* | |x| \text{ is odd }\}.$

4.26 Describe the language generated by the given grammars.

This is a regular grammar. Using the construction given in the proof of Theorem 4.14, we obtain the following NFA accepting $L(G)$.

Now it is not difficult to see that

$L(G) = \{x \in \{a, b\}^* | n_a(x) \text{ is even and } n_b(x) \text{ is odd}\}.$

4.1 In each case say which language is generated by the CFG $G$ with the productions as indicated.

a. $L(G) = a, b^*.$

b. $L(G) = \{a, b\}^* \{a\}.$

c. $L(G) = \{ba\}^* \{b\}.$

d. $L(G) = \{x \in \{a, b\}^* | \text{bb does not occur in } x\}.$

e. $L(G) = \{a, b\}^* \{b\}.$

f. $L(G) = \{x \in \{a, b\}^* | x \text{ is a palindrome over } \{a, b\}\}.$

4.26 Describe the language generated by the given grammars.

a. $S \rightarrow aA | bC | b, \quad A \rightarrow aS | bB, \quad B \rightarrow aC | bA | a, \quad C \rightarrow aB | bS$

This is a regular grammar. Using the construction given in the proof of Theorem 4.14, we obtain the following NFA accepting $L(G)$.

Now it is not difficult to see that

$L(G) = \{x \in \{a, b\}^* | n_a(x) \text{ is even and } n_b(x) \text{ is odd}\}.$

4.1 In each case say which language is generated by the CFG $G$ with the productions as indicated.

a. $L(G) = a, b^*.$

b. $L(G) = \{a, b\}^* \{a\}.$

c. $L(G) = \{ba\}^* \{b\}.$

d. $L(G) = \{x \in \{a, b\}^* | \text{bb does not occur in } x\}.$

e. $L(G) = \{a, b\}^* \{b\}.$

f. $L(G) = \{x \in \{a, b\}^* | x \text{ is a palindrome over } \{a, b\}\}.$

4.26 Describe the language generated by the given grammars.

a. $S \rightarrow aA | bC | b, \quad A \rightarrow aS | bB, \quad B \rightarrow aC | bA | a, \quad C \rightarrow aB | bS$

This is a regular grammar. Using the construction given in the proof of Theorem 4.14, we obtain the following NFA accepting $L(G)$.

Now it is not difficult to see that

$L(G) = \{x \in \{a, b\}^* | n_a(x) \text{ is even and } n_b(x) \text{ is odd}\}.$

4.1 In each case say which language is generated by the CFG $G$ with the productions as indicated.

a. $L(G) = a, b^*.$

b. $L(G) = \{a, b\}^* \{a\}.$

c. $L(G) = \{ba\}^* \{b\}.$

d. $L(G) = \{x \in \{a, b\}^* | \text{bb does not occur in } x\}.$

e. $L(G) = \{a, b\}^* \{b\}.$

f. $L(G) = \{x \in \{a, b\}^* | x \text{ is a palindrome over } \{a, b\}\}.$

4.26 Describe the language generated by the given grammars.

a. $S \rightarrow aA | bC | b, \quad A \rightarrow aS | bB, \quad B \rightarrow aC | bA | a, \quad C \rightarrow aB | bS$

This is a regular grammar. Using the construction given in the proof of Theorem 4.14, we obtain the following NFA accepting $L(G)$.

Now it is not difficult to see that

$L(G) = \{x \in \{a, b\}^* | n_a(x) \text{ is even and } n_b(x) \text{ is odd }\}.$

S corresponds to “even number of $a$’s and even number of $b$’s”

A corresponds to “odd number of $a$’s and even number of $b$’s”

B corresponds to “odd number of $a$’s and odd number of $b$’s”

C corresponds to “even number of $a$’s and odd number of $b$’s”.

b. $S \rightarrow bS | aA | \Lambda, \quad A \rightarrow aA | bB | b, \quad B \rightarrow bS$
This is a regular grammar. Using the construction given in the proof of Theorem 4.14, we obtain the following NFA accepting $L(G)$.

![Diagram of NFA](image)

From this automaton we can read the regular expression $(b^*aa^*bb)^*(\Lambda + b^*aa^*)b$ which describes $L(G)$.

4.27 See the FA $M$ in Figure 4.33. The regular grammar $G$ with $L(G) = L(M)$ constructed from $M$ as in Theorem 4.4 has the productions:

$A \rightarrow aB \mid bD \mid \Lambda,$
$B \rightarrow aB \mid bC \mid b,$
$C \rightarrow aB \mid bC \mid b,$
$D \rightarrow aD \mid bD.$

This grammar has $A$ as its staring symbol. Note that the state $D$ is a 'sink' state and that consequently, the productions relating to $D$ can be safely omitted from the grammar without affecting the successful derivations (and hence the generated language). This yields:

$A \rightarrow aB \mid \Lambda,$
$B \rightarrow aB \mid bC \mid b,$
$C \rightarrow aB \mid bC \mid b.$

4.29 Each of the given grammars, though not regular, generates a regular language. Find for each a regular grammar (a CFG with only productions of the form $X \rightarrow aY$ and $X \rightarrow a$) generating its language.

a. $S \rightarrow SSS \mid a \mid ab$

The only non-terminating production for $S$ is $S \rightarrow SSS$, which means that the number of occurrences of $S$ in the current string increases with 2 each time this production is used. Terminating productions can be postponed until no production $S \rightarrow SSS$ will be applied anymore. Since we begin with one $S$, this means that just before termination we will have an odd number of $S$’s. Termination of $S$ yields for every occurrence of $S$ either $a$ or $ab$. Hence $L(G)$ consists of an odd number of concatenated $a$ or $ab$ strings:

$L(G) = \{a, ab\} \{a, ab\}^* \{a, ab\}$ which is indeed a regular language.

A regular grammar for this language would be (with starting symbol $Z$):

$Z \rightarrow aU \mid aV \mid aAB, \quad B \rightarrow b, \quad V \rightarrow bU. \quad U \rightarrow aZ \mid aW, \quad W \rightarrow bZ$

b. $S \rightarrow AabB, \quad A \rightarrow aA \mid bA \mid \Lambda, \quad B \rightarrow Bab \mid Bb \mid ab \mid b$

It is easy to see that from $A$ the language $\{a, b\}^*$ is generated.

From $B$ we obtain the language $\{ab, b\} \{ab, b\}^* = \{ab, b\}^* \{ab, b\}$.

Consequently $L(G) = \{a, b\}^* \{ab, b\}^* \{ab, b\}$, a regular language.

A regular grammar for this language would be (with starting symbol $Z$):

$Z \rightarrow aZ \mid bZ \mid aB, \quad B \rightarrow bY, \quad Y \rightarrow aX \mid b \mid bY, \quad X \rightarrow b \mid bY$
c. \(S \rightarrow AAS | ab | aab, \quad A \rightarrow ab | ba | \Lambda\)

As long as no terminating productions have been used every string derived from \(S\) consists of an even number of \(A\)’s followed by an \(S\). Upon termination the \(S\) will be rewritten into \(ab\) or \(aab\), while each \(A\) yields \(ab\) or \(ba\) or \(\Lambda\). An even number of concatenated \(A\)’s yields a string consisting of an arbitrary number of concatenated occurrences of \(ab\) and \(ba\). Note that this number is not necessarily even, since any \(A\) may also be rewritten into \(\Lambda\). Consequently, \(L(G) = \{ab, ba\}^*\{ab, aab\}\), a regular language.

A regular grammar for this language would be (with starting symbol \(Z\)):
\[
Z \rightarrow aY | bX,
X \rightarrow aZ,
Y \rightarrow bZ | aW,
W \rightarrow b
\]

d. \(S \rightarrow AB, \quad A \rightarrow aAa | bAb | a | b, \quad B \rightarrow aB | bB | \Lambda\)

From \(A\) we generate the language consisting of all odd-length palindromes over \(\{a, b\}\), which is not a regular language! However \(B\) generates \(\{a, b\}^+\). Thus \(L(G)\) consists of words formed by an odd-length palindrome followed by an arbitrary word over \(\{a, b\}\). Now note that every non-empty word over \(\{a, b\}\) can be seen as an \(a\) or \(b\) (both odd-length palindromes) followed by an arbitrary word over \(\{a, b\}\). Consequently, \(L(G) = \{a, b\}^+\), a regular language after all!

A regular grammar for this (easy) language would be: \(Z \rightarrow aY | bX, \quad X \rightarrow aZ, \quad Y \rightarrow bZ | aW, \quad W \rightarrow b\)

e. \(S \rightarrow AA | B, \quad A \rightarrow AAA | Ab | a | a, \quad B \rightarrow aB | bB | \Lambda\)

Clearly, every occurrence of \(B\) generates \(\{b\}^+\). Because of \(S \rightarrow B\), this implies that \(\{b\}^+ \subseteq L(G)\).

The other production for \(S\) is \(S \rightarrow AA\). Each \(A\) can surround itself with any number of \(b\)’s before either terminating as \(a\) or producing two more \(A\)’s. Hence after \(S \Rightarrow AA\) we can produce any word over \(\{a, b\}\) with an even (non-zero) number of \(a\)’s. Together with \(\{b\}^+ \subseteq L(G)\), this implies that \(L(G) = (\{b\}^*\{a\}^*\{a\}^*\{b\}^*)^+ \cup \{b\}^+\).

A regular grammar for this language would be (with starting symbol \(Z\)):
\[
Z \rightarrow aY | bZ | b, \quad Y \rightarrow bY | aZ | a
\]

4.34 Consider the CFG with productions: \(S \rightarrow a | Sa | bSS | SSb | SbS\). This grammar is ambiguous, the word \(abaa\) has two different leftmost derivations: \(S \Rightarrow SbS \Rightarrow abS \Rightarrow abSa \Rightarrow abaa\) and \(S \Rightarrow Sa \Rightarrow SbSa \Rightarrow abSa \Rightarrow abaa\).

4.36 We look at the grammars given in Exercise 4.1. For each of them we have to decide if the grammar is ambiguous or not. We discuss here b, c, d, e, f and g. Grammars a and h are both not ambiguous, as it can be proved in a similar manner as for grammar g.

b The grammar given in b is ambiguous. This follows from the two different
leftmost derivations for $aaa$:

$$S \Rightarrow SS \Rightarrow SSS \Rightarrow^3 aaa$$

and

$$S \Rightarrow SS \Rightarrow aS \Rightarrow aSS \Rightarrow^2 aaa.$$  

c and d The grammar given c and d are ambiguous. This follows from the two different leftmost derivations for the word $babab$:

$$S \Rightarrow SaS \Rightarrow SaSaS \Rightarrow^3 babab$$

and

$$S \Rightarrow SaS \Rightarrow baS \Rightarrow baSaS \Rightarrow^2 babab.$$  

e This grammar is ambiguous. We have the following two leftmost derivations for $abab$:

$$S \Rightarrow TT \Rightarrow aTT \Rightarrow aTaT \Rightarrow abaT \Rightarrow abab$$

and

$$S \Rightarrow TT \Rightarrow TaT \Rightarrow aTaT \Rightarrow^2 abab.$$  

f First of all note that since all productions have at most one non-terminal at the right hand side, every derivations is a leftmost one.

Next we prove by induction on the length of $x \in \Sigma^*$ that if $S \Rightarrow^* x$ then this is the only derivation of $x$ from $S$, and that if $A \Rightarrow^* x$ then this is the only derivation of $x$ from $A$.

(Induction base) $n = 0$ then $x = \Lambda$. $\Lambda$ is not derivable from $S$ because every production of $S$ introduce a terminal. But $A \Rightarrow^* \Lambda$, because $A \Rightarrow \Lambda$. Clearly this is the only derivation of $\Lambda$ from $A$, because all other productions introduce terminals.

(Induction step) Assume the above statement holds for all strings of length strictly smaller than $x \in \Sigma^*$ such that $S \Rightarrow^* x$ or $A \Rightarrow^* x$.

Assume $S \Rightarrow^* x$. If $x = aya$ then the first step in the derivation of $x$ from $S$ must be $S \Rightarrow aSa$. Thus $S \Rightarrow^* y$. But $y$ is strictly smaller than $x$, and, by induction hypothesis, the derivation $S \Rightarrow^* y$ is unique. Thus also that of $x$ from $S$ is unique. The case when $x = byb$ is similar. If $x = ayb$ then the first step in the derivation of $x$ from $S$ must be $S \Rightarrow aAb$. Thus $A \Rightarrow^* y$, and by induction hypothesis it follows that the latter derivation is unique. And thus so also that of $x$ from $S$ is unique. The case when $x = bya$ is similar.
If \( A \Rightarrow^* x \) we have four cases. The case \( x = aya \) and \( byb \) can be treated as above. If \( x = a \) or \( x = b \) then \( A \Rightarrow x \) is immediately the unique derivation for \( x \) from \( A \). The case \( x = \Lambda \) is not necessary because is treated in the base of the induction.

**g** The proof is similar to f. First we note that since all productions have at most one non-terminal at the right hand side every derivations is a leftmost one. Next we prove by induction on the length of \( x \in \Sigma^* \) that if \( S \Rightarrow^* x \) then this is the only derivation of \( x \) from \( S \), and that if \( T \Rightarrow^* x \) then this is the only derivation of \( x \) from \( T \).

(Induction base) \( n = 0 \) then \( x = \Lambda \). We have that \( S \Rightarrow^* \Lambda \), because \( S \Rightarrow \Lambda \). This is the only derivation of \( \Lambda \) from \( S \), because all other productions introduce terminals. Further, \( \Lambda \) is not derivable from \( T \), because every production of \( T \) introduce a terminal.

(Induction step) Assume the above statement holds for all strings of length strictly smaller than \( x \in \Sigma^* \), with \( S \Rightarrow^* x \) or \( T \Rightarrow^* x \).

Assume \( S \Rightarrow^* x \). If \( x = ay \) then the first step in the derivation of \( x \) from \( S \) must be \( S \Rightarrow aT \). Thus \( T \Rightarrow^* y \). But \( y \) is strictly smaller than \( x \), and, by induction hypothesis, the derivation \( T \Rightarrow^* y \) is unique. Thus also that of \( x \) from \( S \) is unique. The case when \( x = by \) is similar.

Assume \( T \Rightarrow^* x \) If \( x = ay \) then the first step in the derivation of \( x \) from \( T \) must be \( T \Rightarrow aS \). Thus \( S \Rightarrow^* y \). But \( y \) is strictly smaller than \( x \), and, by induction hypothesis, the derivation \( S \Rightarrow^* y \) is unique. Thus also that of \( x \) from \( T \) is unique. The case when \( x = by \) is similar.

**4.38**

We have to show that a given grammar is ambiguous and we have to give a non-ambiguous grammar generating the same language.

**a.** \( S \rightarrow SS \mid a \mid b \)

According to this grammar the string \( aba \) has two different leftmost derivations: \( S \Rightarrow SS \Rightarrow aS \Rightarrow aSS \Rightarrow abS \Rightarrow aba \) and \( S \Rightarrow SS \Rightarrow SSS \Rightarrow aSS \Rightarrow abS \Rightarrow aba \).

The two derivation trees are as follows:

```
      S
     /|
    / S
   /|
  / S
 /|
/ a
b

      S
     /|
    / S
   /|
  / S
 /|
/ a
a
```

```
With the exception of the empty string Λ, all strings over \{a, b\} can be generated, that is the regular language \{a, b\}^*.

An equivalent regular grammar is then: \( S \rightarrow aX \mid bX \quad X \rightarrow aX \mid bX \mid \Lambda \) .

This grammar is not ambiguous because it is a regular grammar stemming from a deterministic finite automaton.

\[ \begin{align*}
S &\rightarrow ABA \\
A &\rightarrow aA \mid \Lambda \\
B &\rightarrow bB \mid \Lambda
\end{align*} \]

According to this grammar, the word \( a \) has two different leftmost derivations:

\[ S \Rightarrow ABA \Rightarrow aABA \Rightarrow a \]

The corresponding derivation trees look like this:

\[ \begin{align*}
S &\quad | \quad \Lambda \\
A &\quad | \quad \Lambda \\
\Lambda &\quad | \quad \Lambda
\end{align*} \]

The grammar generates words of 0 or more \( a \)'s followed by 0 or more \( b \)'s, i.e. the language denoted by the regular expression \( a^* b^* a^* \). An equivalent regular grammar is:

\[ \begin{align*}
S &\rightarrow aS \mid bX \mid \Lambda \\
X &\rightarrow aY \mid A \\
Y &\rightarrow aY \mid \Lambda
\end{align*} \]

This grammar is not ambiguous, because its underlying finite automaton is clearly deterministic. For example, the word \( a \) has the unique derivation \( S \Rightarrow aS \Rightarrow \Lambda \).

\[ \begin{align*}
S &\rightarrow aSb \mid aaSb \mid \Lambda \\
A &\rightarrow aaAb \mid \Lambda
\end{align*} \]

According to this grammar, the word \( aaab \) has two different leftmost derivations:

\[ S \Rightarrow aSb \Rightarrow aaab \Rightarrow aaab \]

The ambiguity of the given grammar is caused by the extra \( a \)'s that can be added at any time. The following grammar generates the same language, but first generates one \( a \) for each \( b \) and if two \( a \)'s per \( b \) are generated, then it proceed so until the derivation stops. Thus, we have an additional non-terminal in order to be able to separate two processes:

\[ \begin{align*}
S &\rightarrow aSb \mid aA \mid aAb \mid \Lambda \\
A &\rightarrow aaAb \mid \Lambda
\end{align*} \]

This grammar in not ambiguous, because the only derivation of each string...
of the form \(a^{2j+k}b^j\), where \(0 \leq k \leq j\), is
\[
S \Rightarrow a^j a^j S b^j \Rightarrow a^j b^j \text{ if } k = 0 \text{ and } \\
S \Rightarrow a^{j-k} a^j a^j S b^j \Rightarrow a^{j-k} \Rightarrow a^{j-k+2}a^{2(k-1)}b^{j-k} \Rightarrow a^{j+k}b^j \text{ if } k \geq 1.
\]

4.48 Let \(G = (V, \Sigma, S, P)\) be a CFG. According to Definition 6.6, a variable is nullable if and only if it has a production with right-hand-side \(\Lambda\) or a production with right-hand-side consisting of nullable variables only. We have to prove that for all \(A \in V\) it holds that \(A\) is nullable if and only if \(A \Rightarrow^* \Lambda\) in \(G\).

Let \(A \in V\). First assume that \(A\) is nullable. We use (structural) induction. If \(A\) is nullable, because of the production \(A \Rightarrow \Lambda\), then we have immediately that \(A \Rightarrow \Lambda\). Otherwise there is a production \(A \Rightarrow B_1B_2\ldots B_n\) with \(n \geq 1\) and the \(B_i\) nullable variables. By the induction hypothesis we have \(B_i \Rightarrow^* \Lambda\) for all \(1 \leq i \leq n\). Thus \(A \Rightarrow B_1B_2\ldots B_n \Rightarrow^* B_2\ldots B_n \Rightarrow^* B_n \Rightarrow^* \Lambda\) as desired.

Next assume that \(A \Rightarrow^m \Lambda\) in \(G\) for some \(m \geq 1\) (the case \(m = 0\) does not occur). We prove by induction on \(m\) that \(A\) is nullable. If \(m = 1\), then \(A \Rightarrow \Lambda\). This implies that \(A \Rightarrow \Lambda\) is a production of \(G\) and so \(A\) is nullable. Next assume (induction hypothesis) that whenever \(B \Rightarrow^k \Lambda\) for some \(k \leq m\), then \(B\) is nullable. Then consider the case \(A \Rightarrow^m \Lambda\). This implies that the first production used in this derivation has been of the form \(A \Rightarrow B_1\ldots B_n\) for some \(n \geq 1\). Thus \(A \Rightarrow B_1\ldots B_n \Rightarrow^* B_2\ldots B_n \Rightarrow^* B_n \Rightarrow^* \Lambda\). Consequently, for each \(1 \leq i \leq n\), we have \(B_i \Rightarrow^k \Lambda\) where \(1 \leq k_i \leq m\). By the induction hypothesis each \(B_i\) is nullable and so also \(A\) is nullable.

4.49 Find a CFG without \(\Lambda\)-productions that generates the same language (except for \(\Lambda\)) as the given CFG. We apply Algorithm 6.1.

a. CFG \(G\) is given as \(S \Rightarrow AB\,|\,A, A \Rightarrow aASb\,|\,a, B \Rightarrow bS\).

The nullable variables are \(N_0 = \{S\} = N_1\).

Modify the productions: \(S \Rightarrow AB\,|\,A, A \Rightarrow aASb\,|\,a, B \Rightarrow bS\,|\,b\).

Finally, remove the \(\Lambda\) productions to obtain \(G'\) with
\[
S \Rightarrow AB, A \Rightarrow aASb\,|\,a, B \Rightarrow bS\,|\,b.
\]

Note that \(S\) is nullable. Thus (see exercise 6.33) \(S \Rightarrow^* \Lambda\) which implies that \(\Lambda \in L(G)\). Hence, in this case \(L(G) - L(G') = \{\Lambda\}\).

b. CFG \(G\) is given as
\[
S \Rightarrow AB\,|\,ABC, A \Rightarrow BA\,|\,BC\,|\,\Lambda\,|\,a, \\
B \Rightarrow AC\,|\,CB\,|\,\Lambda\,|\,b, C \Rightarrow BC\,|\,AB\,|\,A\,|\,c.
\]

The nullable variables are obtained as \(N_3 = N_2 = \{S, A, B, C\}\) from
\(N_0 = \{A, B\}, N_1 = N_0 \cup \{C\}, N_2 = N_1 \cup \{S\}\).
Modify the productions (duplicates not included):

\[ S \rightarrow AB \mid A \mid \Lambda \mid ABC \mid BC \mid AC \mid C, \quad A \rightarrow BA \mid B \mid BC \mid C \mid \Lambda \mid a, \]
\[ B \rightarrow AC \mid A \mid C \mid CB \mid B \mid \Lambda \mid b, \quad C \rightarrow BC \mid B \mid C \mid \Lambda \mid AB \mid A \mid c. \]

Finally, remove the \( \Lambda \) productions and \( X \rightarrow X \) productions to obtain \( G' \)

\[ S \rightarrow AB \mid A \mid \Lambda \mid ABC \mid BC \mid AC \mid C, \quad A \rightarrow BA \mid B \mid BC \mid C \mid a, \]
\[ B \rightarrow AC \mid A \mid C \mid CB \mid b, \quad C \rightarrow BC \mid B \mid AB \mid A \mid c. \]

Note that \( S \) is nullable and so \( \Lambda \in L(G) \). Hence, also in this case \( L(G) - L(G') = \{ \Lambda \} \).
5.1.a \((q_0, ab, Z_0) \vdash (q_1, b, aZ_0) \vdash (q_2, \Lambda, Z_0) \vdash, \text{ acceptance.}\)
\((q_0, aab, Z_0) \vdash (q_1, ab, aZ_0) \vdash (q_2, \Lambda, aZ_0) \vdash, \text{ crash.}\)
\((q_0, abb, Z_0) \vdash (q_1, bb, aZ_0) \vdash (q_2, b, Z_0) \vdash, \text{ crash.}\)
5.1.b \((q_0, bbcbb, Z_0) \vdash (q_0, bcbb, bZ_0) \vdash (q_0, cbb, bbZ_0) \vdash (q_1, b, bZ_0) \vdash (q_1, \Lambda, Z_0) \vdash, \text{ acceptance.}\)
\((q_0, baca, Z_0) \vdash (q_0, aca, bZ_0) \vdash (q_0, ca, abZ_0) \vdash (q_1, a, abZ_0) \vdash (q_1, \Lambda, bZ_0) \not\vdash, \text{ crash.}\)

5.3 As argued in exercise 5.2, the PDA has two options in state \(q_0\): to consider the current symbol as the middle one in an odd length palindrome, or to make a \(\Lambda\)-move to \(q_1\). In addition, there is the option never to leave \(q_0\). Consequently, for an input string of length \(n\) the PDA has \(2^n + 1\) different (complete) computations.

5.4 a. The PDA accepts only even length palindromes once the possibilities (in moves 1-6) to go from \(q_0\) to \(q_1\) while reading an \(a\) or a \(b\) have been removed.

b. The PDA accepts only odd length palindromes once the possibilities (moves 7-9) to go from \(q_0\) to \(q_1\) with a \(\Lambda\)-move have been removed.

5.6 a. This PDA accepts \(\{axa, bxb \mid x \in \{a, b\}^*\}\).

b. The PDA accepts \(\{xcy \mid x, y \in \{a, b\}^* \text{ and } |x| = |y|\}\).

5.10 Let \(M_0 = (Q, \Sigma, q_0, A, \delta)\) be an FA. From \(M_0\) we construct a DPDA \(M\) with two states: \(p_0\) is the initial state; if \(q_0 \in A\), that is \(\Lambda \in L(M_0)\), then \(p_0\) is also the accepting state, otherwise \(p_1\) is the accepting state. The stack alphabet of \(M\) is \(Q\) with \(q_0\) as the initial stack symbol.

The DPDA simulates \(M_0\) as follows: the state on top of the stack corresponds to the current state of \(M_0\). If \(M_0\) moves to a state \(r\) while reading an input symbol \(a\), also \(M\) reads \(a\) and it pushes \(r\) onto the stack while moving to the accepting state if \(r \in A\) and to the non-accepting state if \(r \not\in A\).

Note that \(M\) never removes a symbol from the stack, has no \(\Lambda\)-transitions, and is deterministic (because \(M_0\) is deterministic).

5.28 b. Given is the CFG with productions \(S \rightarrow S + S \mid S \ast S \mid (S) \mid a\).

This grammar generates the string \(x = (a \ast a + a)\). We consider the top-down PDA constructed from the grammar as in Definition 7.4 and trace a sequence of steps by which \(x\) is accepted together with the corresponding leftmost derivation.
Consider the PDA \( M \) from Example 5.7 with move 12 \((q_1, \Lambda, Z_0) = \{(q_2, Z_0)\}\) changed into \((q_1, \Lambda, Z_0) = \{(q_2, \Lambda)\}\).

This PDA accepts the same language \( L = L(M) \) but now by empty stack and so we can apply Theorem 5.29: we consider the CFG as constructed there with \( L(G) = L \).

For \( x = ababa \) the table above shows a sequence of steps in the new PDA by which \( x \) is accepted together with the corresponding leftmost derivation in \( G \).

Let \( M \) be a PDA (accepting with empty stack) and consider the CFG \( G \) obtained from \( M \) as in the proof of Theorem 5.29. Since every accept-
ing computation of $M$ determines a unique leftmost derivation of a word in $L(G)$, it follows that $G$ is unambiguous if $M$ is deterministic. However if $G$ is unambiguous it is not necessarily the case that $M$ is deterministic. It is sufficient if $M$ never has more than one accepting computation per word.

5.38 a Consider the CFG $G$ given by the productions:
$S \rightarrow S_1\$, $S_1 \rightarrow AS_1 | \Lambda$, $A \rightarrow aA | b$.
The top-down PDA associated with $G$ is given below through its transition diagram.

```
$\$, $\$/{\Lambda}
$a$, $a/{\Lambda}$
$b$, $b/{\Lambda}$
$\Lambda$, $S/S_1\$
$\Lambda$, $S_1/AS_1$; $\Lambda$, $S_1/{\Lambda}$
$\Lambda$, $A/aA$; $\Lambda$, $A/b$
```

Note that this PDA is non-deterministic as a consequence of a choice in productions when rewriting $S_1$ or $A$.  

---

25
6.2 Show using the pumping lemma lemma, that the given languages are not context-free.

a. \( L = \{a^i b^j c^k \mid 0 \leq i < j < k \} \).

Suppose \( L \) is a CFL. Then \( L \) satisfies the pumping lemma (Theorem 8.1a).
Let \( n \) be the constant of that lemma. Consider \( u = a^n b^{n+1}c^{n+2} \in L \).
Then \(|u| \geq n\) and thus there exist \( v, w, x, y, z \) such that \( u = vwxyz \) with \(|wy| > 0\),
\(|wxy| \leq n\), and \( vw^i xy^j z \in L \) for every \( i \geq 0 \). We distinguish two cases:

1. \( wy \) contains at least one \( a \). Then, since \(|wxy| \leq n\), there are no \( c \)'s in \( wy \).
   Consequently, \( vw^2 xy^2 z \) contains at least \( n+1 \) \( a \)'s and exactly \( n+2 \) \( c \)'s,
   which implies that \( vw^2 xy^2 z \) is not in \( L \). A contradiction.

2. \( wy \) does not contain any \( a \). Then it must contain a \( b \) or a \( c \). In this case
   \( vw^0 xy^0 z \) contains \( n \) \( a \)'s and either at most \( n \) \( b \)'s or at most \( n+1 \) \( c \)'s.
   Thus \( vw^0 xy^0 z \notin L \), again a contradiction.

Since we get in all (both) cases a contradiction we conclude that the pumping lemma is not satisfied and hence \( L \) is not context-free.

b. \( L = \{xayb \mid x, y \in \{a, b\}^* \text{ and } |x| = |y| \} \) is a CFL; give a grammar.

\[
S \rightarrow Yb \quad Y \rightarrow aSX \mid bSX \mid a \quad X \rightarrow a|b.
\]
c. \( L = \{ xcx \mid x \in \{ a, b \}^* \} \) is not a CFL; proof similar as in Example 6.2.

d. \( L = \{ xyx \mid x, y \in \{ a, b \}^* \text{ and } |x| \geq 1 \} \) is not a CFL; 
Assume that \( L \) is context-free. Then it satisfies the pumping lemma. Let \( n \) 
be the constant of that lemma. Now consider \( u = xyx \) with \( x = a^n b^n \) and 
\( y = \Lambda \). Thus \( u = a^n b^n a^n b^n \). There must exist words \( p, q, r, s, t \) 
such that \( u = pqrst \) such that \(|qs| > 0, |qrs| \leq n, \) and \( p^i q r^i s t \in L \) for every \( i \geq 0 \). We 
distinguish two cases:

1. \( qs \) consists only of \( a \)'s from the first group of \( a \)'s in \( u \) or it consists only 
of \( b \)'s from the second group of \( b \)'s. Then \( p q^2 r s^2 t \) is either \( a^{n+j} b^n a^n b^n \) 
or \( a^n b^n a^j b^{n-j} \) for some \( j \geq 1 \), which are both not in \( L \). A contradiction.

2. \( qs \) contains a \( b \) from the first group of \( b \)'s in \( u \) or it contains an \( a \) from the 
second group of \( a \)'s in \( u \). Then \( p q^0 r s^0 t \) is either \( a^k b^l a^m b^m \) or \( a^n b^k a^l b^m \)
with \( k, m \geq 1 \) and \( l < n \). Neither of these words is in \( L \), again a contradiction.

Consequently, we always end up with a contradiction and so \( L \) is not a CFL.

6.12 b \( L \) is not a CFL and \( F \) is a finite language. Then \( L - F \) is not a CFL, 
which we prove by contradiction. Assume that \( L - F \) is a CFL.

Note that \( L \cap F \subseteq F \) is finite and hence a CFL. Since a union of two CFLs is 
a CFL, it follows that \( (L - F) \cup (L \cap F) = L \) is context-free, a contradiction.
We conclude that \( L - F \) is not a CFL.

6.13

a. Let \( L \) be a CFL and \( F \) a regular language. Then \( L - F = L \cap \overline{F} \) is a 
CFL, because the complement \( \overline{F} \) of a regular language is regular, and the 
intersection of a CFL with a regular language is a CFL.

b. \( L \) is not a CFL and \( F \) is a regular language. Then \( L - F \) may or may 
not be a CFL. As seen above in 6.12b , if \( F \) is a finite language, then \( L - F \) 
is not context-free. On the other hand, if we let \( F = \Sigma^* \) where \( \Sigma \) is an 
alphabet such that \( L \subseteq \Sigma^* \), then \( L - F = \emptyset \) which is a CFL.