# Automata Theory

Answer to selected exercises 4
John C Martin: Introduction to Languages and the Theory of Computation
Fourth edition

Rudy van Vliet (and Marcello Bonsangue and Jetty Kleijn)

Fall 2024

**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.** $L(G) = \{x \in \{a, b\}^* \mid bb \text{ does not occur in } x\}$.
**e.** $L(G) = \{a\}^*\{b\}\{a\}^*\{b\}\{a\}^*$, i.e. the language of all words with exactly two occurrences of $b$.
**f.** $L(G) = \{xaybx^r, xbyax^r \mid x, y \in \{a, b\}^* \wedge y = y^r\}$, i.e. the language of all words which are palindromes over $\{a, b\}$ with exactly one single "mistake".
**g.** $L(G) = \{x \in \{a, b\}^* \mid |x| \text{ is even }\}$.
**h.** $L(G) = \{x \in \{a, b\}^* \mid |x| \text{ is odd }\}$.

**4.3** Find a context-free grammar generating the given language.
**a.** For $L = \{xay \mid x, y \in \{a, b\}^* \wedge |x| = |y|\}$, the CFG with productions
$S \to aSa \mid aSb \mid bSa \mid bSb \mid a$
**b.** For $L = \{xaay, xbby \mid x, y \in \{a, b\}^* \wedge |x| = |y|\}$, the CFG with
$S \to aSa \mid aSb \mid bSa \mid bSb \mid aa \mid bb$
**c.** For $L = \{a, b\} \cup \{axaya, bxbyb \mid x, y \in \{a, b\}^* \wedge |x| = |y|\}$, the CFG with
$S \to a \mid b \mid aAa \mid bBb,\ \ A \to aAa \mid aAb \mid bAa \mid bAb \mid a,\ \ B \to aBa \mid aBb \mid bBa \mid bBb \mid b$

**4.4** The productions of two context-free grammars are given. Prove that neither one generates the language $L = \{x \in \{a, b\}^* \mid n_a(x) = n_b(x)\}$, the language consisting of all words with an equal number of a's and b's.
**a.** $S \to SabS \mid SbaS \mid \Lambda$
Clearly, every word generated by this grammar has an equal number of a's

and b's, but it cannot generate every word of $L$: Every non-empty word generated by this grammar is of the form $xaby$ or $xbay$ with both $x$ and $y$ also generated by $S$. Hence if $x$ (or $y$) is non-empty it also contains at least one occurrence of $ab$ or $ba$. This implies that $aabb$ cannot be generated even though it is in $L$.
**b.** $S \to aSb \,|\, bSa \,|\, abS \,|\, baS \,|\, Sab \,|\, Sba \,|\, \Lambda$
Clearly, every word generated by this grammar has an equal number of a's and b's, but it cannot generate every word of $L$: Every non-empty word generated by this grammar is of the form $ayb$, $bya$, $aby$, $bay$, $yab$, or $yba$ with $y$ also a word generated by the grammar. Consequently, $x = aabbbbaa$ cannot be generated even though $x \in L$.

**4.5** $S \to aSbScS \,|\, aScSbS \,|\, bSaScS \,|\, bScSaS \,|\, cSaSbS \,|\, cSbSaS \,|\, \Lambda$ .
Does the CFG $G$ with these productions generate the language
$L = \{x \in \{a, b, c\}^* \mid n_a(x) = n_b(x) = n_c(x)\}$?
No. Since every production introduces an equal number of $a$'s, $b$'s, and $c$'s, it is clear that $L(G)$ is included in $L$. Thus the question is whether $G$ can generate *every* word in $L$. This turns out to be not the case.
Consider $aabbcc \in L$. Any derivation of this word has to start with an application of the production $S \to aSbScS$ because we need an $a$ in the first place and $b$'s have to precede $c$'s. To derive $aabbcc$ from $aSbScS$, rewriting the first occurrence of $S$ should lead to the terminal word $a$ or $ab$, but this is impossible, because each word derivable from $S$ has an equal number of $a$'s, $b$'s, and $c$'s (as observed before).

- Give a CFG that generates all regular expressions over an alphabet $\Sigma$.

For simplicity, let us assume $\Sigma = \{a, b\}$. It is easy to generalize the result below to other aplphabets. De terminal symbols includes all elements of $\Sigma$, the operators $\{+, \cdot, ^*, (, )\}$, and distinct symbols for $\emptyset$ en $\Lambda$, say $\phi$ en $\lambda$, respetively. We construct a grammar with starting symbol $S$ and with the following productions:

$$S \;\to\; (S + S) \mid (S \cdot S) \mid (S^*) \mid a \mid b \mid \lambda \mid \phi \;.$$

**4.10** Find a CFG for each of the given langages.
**a.** $S \;\to\; aSb \mid B$ and $B \;\to\; bB \mid \Lambda$.
**b.** $S \;\to\; aSb \mid B$ and $B \;\to\; bB \mid b$.
**c.** $S \;\to\; aSbb \mid \Lambda$.
**d.** $S \;\to\; aSb \mid aSbb \mid \Lambda$.

**e.** $S \rightarrow aSBB \mid \Lambda$ and $B \rightarrow b \mid \Lambda$.
**f.** $S \rightarrow aSBB \mid a \mid ab$ and $B \rightarrow b \mid \Lambda$.

- Find a CFG for each of the given langages.

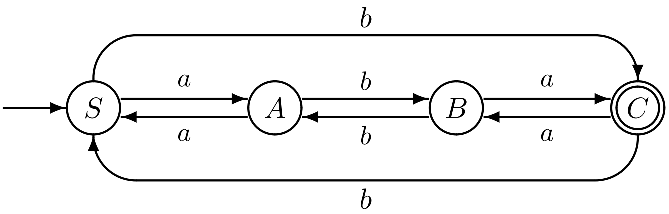
**a.** $L = \{a^i b^j c^k \mid i = j + k\}$. Thus each word in $L$ has the form $a^k a^j b^j c^k$ and such words are exactly generated by the CFG with productions
$S \rightarrow aSc \,|\, T, \quad T \rightarrow aTb \,|\, \Lambda.$
**e.** $L = \{a^i b^j c^k \mid i < j \vee i > k\}$. Thus each word in $L$ is of the form $a^i b^i b^n c^k$ or $a^k a^n b^j c^k$ for some $n \geq 1$. Such words are exactly generated by the CFG with productions
$S \rightarrow XC \,|\, A$
$X \rightarrow aXb \,|\, Xb \,|\, b, \quad C \rightarrow Cc \,|\, \Lambda,$
$Y \rightarrow aYc \,|\, aY \,|\, aZ, \quad Z \rightarrow bZ \,|\, \Lambda.$
**h.** $L = \{a^i b^j \mid i \leq j \leq 2i\}$. Thus each word in $L$ is in $\{a\}^i \{b, bb\}^i$ for some $i \geq 0$. These words are exactly generated by the CFG with productions
$S \rightarrow aSb \,|\, aSbb \,|\, \Lambda.$

**4.25** Given a language $L \subseteq \Sigma^*$ we need to prove that *a.*, *b.* and *c.* are equivalent.
**a.** implies **b.**: it follows directly because regular grammars are a special case of the grammars specified in *b.*
**b.** implies **a.**: Let $L$ be a language generated by a grammar with productions of the form $A \rightarrow xB$ or $A \rightarrow \Lambda$ with $A, B$ variables and $x \in \Sigma^*$. First we find an equivalent grammar without unit productions (i.e. without productions $A \rightarrow xB$ with $|x| = 0$) using Theorem 4.28. In the resulting grammar, we look at all its productions. If $A \rightarrow xB$ with $|x| = 1$ we leave it as is, but if $x = a_1 a_2 \cdots a_n$ for $n \geq 2$ and each $a_i \in \Sigma$, then we substitute $A \rightarrow xB$ by a sequence of productions $A \rightarrow a_1 X_1$, $X_1 \rightarrow a_2 X_2$, ..., $X_{n-2} \rightarrow a_{n-1} X_{n-1}$, $X_{n-1} \rightarrow a_n B$, with $X_1, \ldots, X_{n-1}$ new variable symbols.

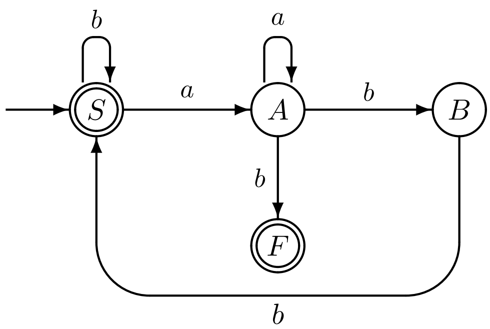
The new grammar so obtained is clearly regular and generates the same language as the original grammar.
**a.** implies **c.**: Assume $L$ is regular. Because the regular languages are closed under reversal, the language $L^r = \{y^r \mid y \in L\}$, where $y^r$ is the reverse of $y$. is also regular. Hence, there exists a regular grammar for $L^r$. Now, it is enough to change every production $A \rightarrow \sigma B$ in this grammar into $A \rightarrow B\sigma$. The language of the resulting grammar is the reverse of $L^r$, and thus equal to $L$. Clearly, this grammar is a special case of the grammar specified in **c.**
**c.** implies **a.**: We first transform each production $A \rightarrow Bx$ into $A \rightarrow x^r B$, where $x^r$ is the reverse of $x$. The new grammar generates $L^r = \{y^r \mid y \in L\}$. Because this grammar is of the form as specified in **b.**, its language $L^r$ is

regular (because **b.** implies **a.**). Since regular language are closed under reversal, the original language $L$ must be regular, too.

**4.26** Draw NFAs accepting the languages generated by the given grammars.
**a.** $S \to aA \mid bC, \quad A \to aS \mid bB, \quad B \to aC \mid bA, \quad C \to aB \mid bS \mid \Lambda$
This is a regular grammar. Using the construction given in the proof of Theorem 4.14, we obtain the following NFA accepting $L(G)$.



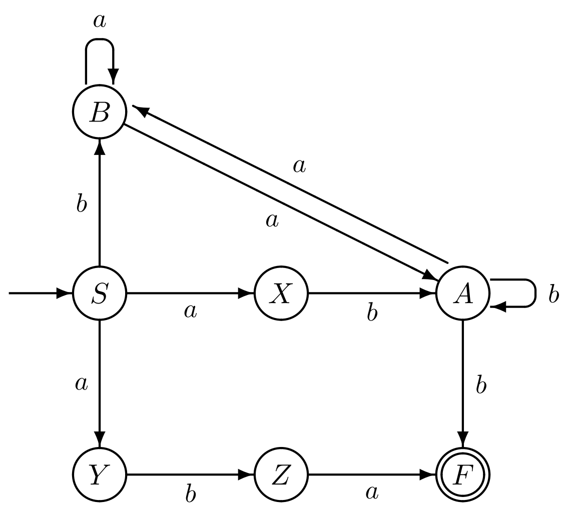

Now it is not difficult to see that
$L(G) = \{x \in \{a, b\}^* \mid n_a(x) \text{ is even and } n_b(x) \text{ is odd }\}$.
$S$ corresponds to "even number of $a$'s and even number of $b$'s"
$A$ corresponds to "odd number of $a$'s and even number of $b$'s"
$B$ corresponds to "odd number of $a$'s and odd number of $b$'s"
$C$ corresponds to "even number of $a$'s and odd number of $b$'s".
**b.** $S \to bS \mid aA \mid \Lambda, \quad A \to aA \mid bB \mid b, \quad B \to bS$
In principle, we again use the construction given in the proof of Theorem 4.14, to obtain an NFA accepting $L(G)$. Only the production $A \to b$ does not satisfy the definition of a regular grammar. For that production, we introduce a special accepting state $F$, reachable from state $A$ with a transition labelled by $b$, and without outgoing transitions. The result is:




From this automaton we can read the regular expression $(b^*aa^*bb)^*b^*(\Lambda + aa^*b)$ 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 4.14 has the productions:
$A \to aB \,|\, bD \,|\, \Lambda, \quad B \to aB \,|\, bC, \quad C \to aB \,|\, bC \,|\, \Lambda, \quad D \to aD \,|\, bD.$
This grammar has $A$ as its starting 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:
$A \to aB \,|\, \Lambda, \quad B \to aB \,|\, bC, \quad C \to aB \,|\, bC \,|\, \Lambda.$

**4.28** Given is the CFG with productions:
$S \to abA \,|\, bB \,|\, aba, \quad A \to b \,|\, aB \,|\, bA, \quad B \to aB \,|\, aA.$
This grammar is not a regular grammar but we transform it into an equivalent regular CFG $G$:
$S \to aX \,|\, bB \,|\, aY, \quad X \to bA, \quad Y \to bZ, \quad Z \to aF,$
$A \to bF \,|\, aB \,|\, bA, \quad B \to aB \,|\, aA, \quad F \to \Lambda.$
Next we apply the method from the proof of Theorem 4.14 and obtain an NFA accepting $L(G)$:




**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 $X \to aY$ and $X \to \Lambda$) generating its language.
**a.** $S \to SSS \,|\, a \,|\, ab$
The only non-terminating production for $S$ is $S \to SSS$, 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 $S \to SSS$ 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 $Z$):
$Z \to aU \,|\, aV \,|\, aF \,|\, aB, \quad B \to bF, \quad V \to bU, \quad U \to aZ \,|\, aW, \quad W \to bZ,$
$F \to \Lambda$
**b.** $S \to AabB, \quad A \to aA \,|\, bA \,|\, \Lambda, \quad B \to Bab \,|\, Bb \,|\, ab \,|\, b$
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 $L(G) = \{a, b\}^*\{ab\}\{ab, b\}^*\{ab, b\}$, a regular language.
A regular grammar for this language would be (with starting symbol $Z$):
$Z \to aZ \,|\, bZ \,|\, aB, \quad B \to bY, \quad Y \to aX \,|\, bF \,|\, bY, \quad X \to bF \,|\, bY, \quad F \to \Lambda$
**c.** $S \to AAS \,|\, ab \,|\, aab, \quad A \to ab \,|\, ba \,|\, \Lambda$
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 $Z$):
$Z \to aY \,|\, bX, \quad X \to aZ, \quad Y \to bZ \,|\, bF \,|\, aW, \quad W \to bF, \quad F \to \Lambda$

**d.** $S \to AB, \quad A \to aAa \,|\, bAb \,|\, a \,|\, b, \quad B \to aB \,|\, bB \,|\, \Lambda$
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 (with starting symbol $Z$): $Z \to aZ \,|\, bZ \,|\, aF \,|\, bF, \quad F \to \Lambda$
**e.** $S \to AA \,|\, B, \quad A \to AAA \,|\, Ab \,|\, bA \,|\, a, \quad B \to bB \,|\, b$
Clearly, every occurrence of $B$ generates $\{b\}^+$. Because of $S \to B$, this implies that $\{b\}^+ \subseteq L(G)$.
The other production for $S$ is $S \to AA$. Each $A$ can surround itself with

any number of $b$'s before either terminating as $a$ or producing two more $A$'s. Eventually, each $A$ from $S \to AA$, yields an odd number of $a$'s, together with an arbitrary number of $b$'s at arbitrary positions. 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 $L(G) = (\{b\}^*\{a\}\{b\}^*\{a\}\{b\}^*)^+ \cup \{b\}^+$, i.e., all non-empty strings with an even number of $a$'s.
A regular grammar for this language would be (with starting symbol $Z$):
$Z \to aY \mid bZ \mid bF, \quad Y \to bY \mid aZ \mid aF, \quad F \to \Lambda$

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

**4.35** Consider the context-free grammar with productions
$S \to AB, \quad A \to aA \mid \Lambda, \quad B \to ab \mid bB \mid \Lambda$
This grammar is NOT unambiguous, even though every derivation of a string from $S$ has to begin with $S \to AB$, and any string derivable from A has only one derivation from $A$ and likewise for $B$.
There are strings in $L(G)$ which have more than one derivation tree (more than one leftmost derivation). Examples are $ab$ and $aab$:
$S \Rightarrow AB \Rightarrow B \Rightarrow ab$ and $S \Rightarrow AB \Rightarrow aAB \Rightarrow aB \Rightarrow abB \Rightarrow ab$;
$S \Rightarrow AB \Rightarrow aAB \Rightarrow aB \Rightarrow aab$ and $S \Rightarrow AB \Rightarrow aAB \Rightarrow aaAB \Rightarrow aaB \Rightarrow aabB \Rightarrow aab$.

**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 derivation 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.

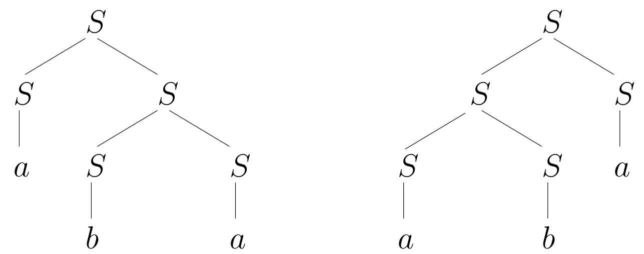
If $A \Rightarrow^* x$ we have four cases. The case $x = aya$ and $byb$ can be treated as above. If $x = a$ or $x = b$ then $A \Rightarrow x$ is immediately the unique derivation for $x$ from $A$. The case $x = \Lambda$ is not necessary because is treated in the base of the induction.
**g** The proof 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 $T \Rightarrow^* x$.

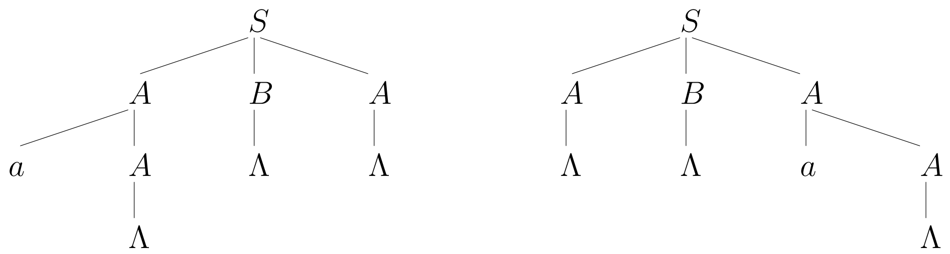
Assume $S \Rightarrow^* x$. If $x = ay$ then the first step in the derivation of $x$ from $S$ must be $S \Rightarrow aT$. Thus $T \Rightarrow^* y$. But $y$ is strictly smaller than $x$, and, by induction hypothesis, the derivation $T \Rightarrow^* y$ is unique. Thus also that of $x$ from $S$ is unique. 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.** $S \to SS \,|\, a \,|\, b$
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:




With the exception of the empty string $\Lambda$, all strings over $\{a, b\}$ can be generated, that is the regular language $\{a, b\}^+$.
An equivalent regular grammar is then : $S \to aX \,|\, bX \quad X \to aX \,|\, bX \,|\Lambda$ .
This grammar is not ambiguous because it is a regular grammar stemming from a deterministic finite automaton.
**b.** $S \to ABA \quad A \to aA \,|\, \Lambda \quad B \to \,|\, bB \,|\, \Lambda$
According to this grammar , the word $a$ has two different leftmost derivations : $S \Rightarrow ABA \Rightarrow aABA \Rightarrow^3 a$ and
$S \Rightarrow ABA \Rightarrow BA \Rightarrow A \Rightarrow aA \Rightarrow a$.
The corresponding derivation trees look like this:




The grammar generates words of 0 or more $a$'s followed by 0 or more $b$'s followed by 0 or more $a$'s, i.e., the language denoted by the regular expression $a * b * a*$. An equivalent regular grammar is:
$S \to aS \,|\, bX \,|\, \Lambda \quad X \to bX \,|\, aY \,|\, \Lambda \quad Y \to aY \,|\, \Lambda$ .
This grammar is not ambiguous, because its underlying finite automaton is clearly deterministic. For example, the word $a$ has the unique derivation $S \Rightarrow aS \Rightarrow \Lambda$.
**c.** $S \to aSb \,|\, aaSb \,|\, \Lambda$ .
According to this grammar , the word $aaab$ has two different leftmost derivations : $S \Rightarrow aSb \Rightarrow aaaSb \Rightarrow aaab$ and
$S \Rightarrow aaSb \Rightarrow aaaSb \Rightarrow aaab$ .
The grammar generates the words consisting of a number of $a$'s followed by a number of $b$'s where the number of $a$'s is at least as large as the number of $b$'s but no more than twice as large, i.e, the language $\{a^i b^j \mid j \le i \le 2j\}$. 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 once two $a$'s for a $b$ are generated, the grammar continues to do so until the derivation stops. Thus, we have an additional non-terminal in order to be able to separate two processes:
$S \to aSb \,|\, \Lambda \,|\, aaAb \quad A \to aaAb \,|\, \Lambda$ .
This grammar in not ambiguous, because the only derivation of each string of the form $a^{j+k}b^j$, where $0 \le k \le j$, is
$S \Rightarrow^j a^j S b^j \Rightarrow a^j b^j$ if $k = 0$ and
$S \Rightarrow^{j-k} a^{j-k} S b^{j-k} \Rightarrow a^{j-k} aaAb b^{j-k} \Rightarrow^{k-1} a^{j-k+2} a^{2(k-1)} A b^{k-1} b^{j-k+1} \Rightarrow a^{j+k} b^j$ if $k \ge 1$.

**4.39** Let $G$ be a regular grammar (note that $\Lambda \notin L(G)$). Convert $G$ into an NFA $M_G$ as in the proof of Theorem 4.14. Make $M_G$ deterministic (using the subset construction) and transform the resulting FA $M$ in an equivalent unambiguous regular grammar.

- *Let $G$ be a context-free grammar with start variable $S$ and the following* *productions:*

$$S \to aSbS \ \mid \ bSaS \ \mid \ \Lambda$$

*This grammar generates* AEqB $= \{x \in \{a,b\}^* \ \mid \ n_a(x) = n_b(x)\}$ *and is ambiguous.*

**c.** *Give an unambiguous context-free grammar for* AEqB*.*

An element $x$ of $AEqB$ is either $\Lambda$, or starts with $a$ or starts with $b$. If $x$ starts with $a$, then there must be a corresponding $b$, i.e., an occurrence of $b$ that causes the number of $a$'s and the number of $b$'s in the current prefix of $x$ to be equal, *for the first time* (after the starting $a$). Let us write $x = aybz$, where $y$ is the substring between the starting $a$ and its corresponding $b$. This substring $y$ is an element of $AEqB$ and has the additional property that each prefix of $y$ has at least as many $a$'s as $b$'s. Such substrings $y$ can be generated by the following context-free grammar:

$$S_a \to aS_abS_a \mid \Lambda$$

Analogously, strings $y$ in $AEqB$ with the additional property that each prefix of $y$ has at least as many $b$'s as $a$'s can be generated by the following context-free grammar:

$$S_b \to bS_baS_b \mid \Lambda$$

To generate $AEqB$, we add the following productions:

$$S \to aS_abS \mid bS_baS \mid \Lambda$$

**4.48** Let $G = (V, \Sigma, S, P)$ be a CFG. According to Definition 4.26, a variable is nullable if and only if it has a production with righthand-side $\Lambda$ or a production with righthand-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 $A \to \Lambda$, then we have immediately that $A \Rightarrow \Lambda$. Otherwise there is a production $A \to B_1B_2 \ldots B_n$ with $n \geq 1$ and all $B_i$'s nullable variables. Assume that for $1 \leq i \leq n$, we indeed have $B_i \Rightarrow^* \Lambda$ (induction hypothesis). Then by the induction hypothesis, $A \Rightarrow B_1B_2 \ldots B_n \Rightarrow^* B_2 \ldots B_n \Rightarrow^* B_n \Rightarrow^* \Lambda$ 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 $A \to \Lambda$ is a production of $G$ and so $A$ is nullable. Let $k \geq 1$ and assume that whenever $B \Rightarrow^m \Lambda$ for some $m \leq k$, then $B$ is

nullable (induction hypothesis). Then consider the case $A \Rightarrow^{k+1} \Lambda$. This implies that the first production used in this derivation has been of the form $A \to B_1 \dots B_n$ for some $n \geq 1$. Thus $A \Rightarrow B_1 \dots B_n \Rightarrow^k \Lambda$. Consequently, for each $1 \leq i \leq n$, we have $B_i \Rightarrow^{k_i} \Lambda$ where $1 \leq k_i \leq k$. 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 the algorithm from Theorem 4.27.
**a.** CFG $G$ is given as $S \to AB \mid \Lambda, \quad A \to aASb \mid a, \quad B \to bS.$
Starting from $N_0 = \emptyset$, we find that the nullable variables are $N_1 = \{S\} = N_2$.
Modify the productions: $S \to AB \mid \Lambda, \quad A \to aASb \mid aAb \mid a, \quad B \to bS \mid b.$
Finally, remove the $\Lambda$ productions to obtain $G'$ with
$S \to AB, \quad A \to aASb \mid aAb \mid a, \quad B \to bS \mid b.$
Note that $S$ is nullable. Thus (see Exercise 4.48) $S \Rightarrow^* \Lambda$ which implies that $\Lambda \in L(G)$. Hence, in this case $L(G) - L(G') = \{\Lambda\}$.
**b.** CFG $G$ is given as
$S \to AB \mid ABC, \quad A \to BA \mid BC \mid \Lambda \mid a,$
$B \to AC \mid CB \mid \Lambda \mid b, \quad C \to BC \mid AB \mid A \mid c.$
The nullable variables are obtained as $N_3 = N_2 = \{S, A, B, C\}$ from
$N_0 = \emptyset$, $N_1 = \{A, B\}$, $N_2 = N_1 \cup \{S, C\}$.
Modify the productions (duplicates not included):
$S \to AB \mid A \mid B \mid \Lambda \mid ABC \mid BC \mid AC \mid C, \quad A \to BA \mid B \mid A \mid BC \mid C \mid \Lambda \mid a,$
$B \to AC \mid A \mid C \mid CB \mid B \mid \Lambda \mid b, \quad C \to BC \mid B \mid C \mid \Lambda \mid AB \mid A \mid c.$
Finally, remove the $\Lambda$ productions and $X \to X$ productions to obtain $G'$
$S \to AB \mid A \mid B \mid ABC \mid BC \mid AC \mid C, \quad A \to BA \mid B \mid BC \mid C \mid a,$
$B \to AC \mid A \mid C \mid CB \mid b, \quad C \to BC \mid B \mid AB \mid A \mid c.$
Note that $S$ is nullable and so $\Lambda \in L(G)$. Hence, also in this case $L(G) - L(G') = \{\Lambda\}$.

**4.50** For each grammar $G$ given, find a CFG $G'$ without $\Lambda$-productions and without unit productions such that $L(G') = L(G) - \{\Lambda\}$. We apply Theorem 4.27 (Note that eliminating $\Lambda$-productions may introduce new unit productions, whereas eliminating unit productions does not introduce $\Lambda$-productions.)
**a.** $G$ has productions $S \to ABA, \quad A \to aA \mid \Lambda, \quad B \to bB \mid \Lambda.$
Elimination of nullable productions: all variables of $G$ are nullable, because $N_3 = N_2 = N_1 \cup \{S\}$ with $N_1 = \{A, B\}$.
Modifying the productions leads to
$S \to ABA \mid BA \mid AA \mid AB \mid B \mid A \mid \Lambda, \quad A \to aA \mid a \mid \Lambda, \quad B \to bB \mid b \mid \Lambda.$
Then we delete the $\Lambda$-productions and we obtain:

$S \to ABA \,|\, BA \,|\, AA \,|\, AB \,|\, B \,|\, A, \quad A \to aA \,|\, a, \quad B \to bB \,|\, b.$
Elimination of unit productions: Both $A$ and $B$ are $S$-derivable; since neither $A$ nor $B$ have unit productions, there are no variables that are $A$-derivable or $B$-derivable.
$A$ is $S$-derivable, so we add $S \to aA$ and $S \to a$;
$B$ is $S$-derivable, so we add $S \to bB$ and $S \to b$.
Then we delete all unit productions.
Consequently we arrive at the CFG $G'$ defined by
$S \to ABA \,|\, BA \,|\, AA \,|\, AB \,|\, bB \,|\, b \,|\, aA \,|\, a, \quad A \to aA \,|\, a, \quad B \to bB \,|\, b.$

**4.51, 4.52, 4.53** These exercises are all concerned with reducing CFGs in the sense that superfluous symbols (those that can never be used in a successful derivation) are removed. Let $G = (V, \Sigma, S, P)$ be a CFG.

**4.51** Live variables:
$A$ is live (in $G$) iff there exists an $x \in \Sigma^*$ such that $A \Rightarrow^* x$.
Recursive definition/algorithm:
$N_0 = \emptyset$,
$N_{i+1} = N_i \cup \{A \in V \mid \exists x \in (N_i \cup \Sigma)^* \text{ with } A \to x \in P\}$ for all $i \geq 0$.
In particular, $N_1 = \{A \in V \mid \exists x \in \Sigma^* \text{ with } A \to x \in P\}$.
The algorithm terminates if $N_{k+1} = N_k$ for some $k \geq 0$.

**4.52** Reachable variables:
$A$ is reachable (in $G$) iff there exists $\alpha, \beta \in (V \cup \Sigma)^*$ such that $S \Rightarrow^* \alpha A \beta$.
Recursive definition/algorithm:
$N_0 = \{S\}$ and, for all $i \geq 0$,
$N_{i+1} = N_i \cup \{A \in V \mid \exists B \in N_i \text{ for which } \exists \alpha_1, \alpha_2 \in (V \cup \Sigma)^* \text{ with } B \to \alpha_1 A \alpha_2 \in P\}$.
The algorithm terminates if $N_{k+1} = N_k$ for some $k \geq 0$.

**4.53** Useful variables:
$A$ is useful (in $G$) iff there exists $\alpha, \beta \in (V \cup \Sigma)^*$ and $x \in \Sigma^*$ such that $S \Rightarrow^* \alpha A \beta \Rightarrow^* x$. Thus if $A$ is useful, it is reachable and live.
**c.i.** Note that only useful variables appear in successful derivations (and vice versa: each useful variable appears in some successful derivation). As discussed in **a.** we can find for each CFG an equivalent CFG in which all variables are useful by first eliminating all dead variables and then all non-reachable ones. As an example consider the grammar $G$ given by the productions
$S \to ABC \,|\, BaB, \quad A \to aA \,|\, BaC \,|\, aaa, \quad B \to bBb \,|\, a, \quad C \to CA \,|\, AC.$

First determine the live variables: $N_0 = \emptyset$, $N_1 = \{A, B\}$, $N_2 = N_1 \cup \{S\}$, $N_3 = N_2$.
Eliminate the remaining ("dead") variables (in this case $C$) from $G$:
$S \to BaB, \quad A \to aA \,|\, aaa, \quad B \to bBb \,|\, a.$
Next determine (in the new grammar) the reachable variables: $N_0 = \{S\}$, $N_1 = N_0 \cup \{B\}$, $N_2 = N_1$.
Eliminate the remaining, unreachable, variables (in this case $A$) from $G$:
$S \to BaB, \quad B \to bBb \,|\, a.$
This grammar generates $L(G)$ and is "reduced" (all its variables are useful).
Finally, note that eliminating dead variables may make others unreachable: For the example just worked out, eliminating $S \to ABC$ (because $C$ is dead) makes $A$ unreachable. On the other hand, eliminating non-reachable variables does not affect the liveness of the (reachable) others.

**4.54** Construct for each grammar $G$ given, a grammar $G'$ in CNF with $L(G') = L(G) - \{\Lambda\}$.
**a.** $G$ with productions $S \to SS \,|\, (S) \,|\, \Lambda$.
1. Eliminate the $\Lambda$-production from $G$ which yields $G_1$ with productions $S \to SS \,|\, (S) \,|\, ()$. The newly introduced production $S \to S$ is removed together with the $\Lambda$-production. $L(G_1) = L(G) - \{\Lambda\}$.
2. There are no unit productions (left).
3. Finally, adapt to CNF; first we get $S \to SS \,|\, LSR \,|\, LR, \quad L \to (, \quad R \to )$;
next we have $S \to SS \,|\, LX \,|\, LR, \quad X \to SR, \quad L \to (, \quad R \to )$,
which are the productions of $G'$ and $L(G') = L(G_1) = L(G) - \{\Lambda\}$.

• Let $G = (V, \Sigma, S, P)$ be a CFG in Chomsky normal form and $x \in L(G)$ with $|x| = k$ for some $k \geq 1$. We compute the number of derivation steps needed to generate $x$.
As in the beginning of Section 4.5, we consider, for words $w \in (V \cup \Sigma)^*$, their length $|w|$ and the number of occurrences of terminals which appear in them: $t(w)$. Let $N(w) = |w| + t(w)$. Thus $N(S) = 1$ and $N(x) = |x| + t(x) = k + k = 2k$ for our given $x$. Since $G$ is in CNF its productions are of the form $A \to BC$ or $A \to a$. Consequently, applying a production in a single derivation step $u \Rightarrow v$ either increases the length by 1 or increases the number of terminal occurrences by 1. In other words: $N(v) = N(u) + 1$. Since $N(x) - N(S) = 2k - 1$, it follows that a (each!) derivation of $x$ from $S$ in $G$ consists of $2k - 1$ derivation steps.

---

Version of 11 November 2024. Feel free to mention any errors in these solutions at rvvliet@liacs.nl