Opgaven uit oude tentamens FI1.

1. De taal K bestaat uit alle woorden (over $\{a, b\}$) met precies één voorkomen van deelwoord ab, dus $a\underline{ab}aa \in K$, maar $a\underline{ab}a\underline{ab} \notin K$.

   a. Laat zien dat K^2 *niet* de taal is van alle woorden met precies twee voorkomens van ab.

   We kijken naar het complement van K.

   b. Geef een deterministische eindige automaat voor de taal $\{a, b\}^* - K$.

   c. Laat zien dat $\{a, b\}^* - K$ regulier is, dat wil zeggen druk de taal uit in eindige talen met behulp van de operaties vereniging, concatenatie en ster.

2. Gegeven is de taal $K =$

$$\{\ w \in \{a, b\}^* \mid n_a(w) \text{ is even, en elke } a \text{ wordt gevolgd door een } b\ \}$$

   Voorbeelden:
   $\lambda \in K$, $bbbbbb \in K$, $abbbabbb \in K$, $bababaab \notin K$, $babbabbbab \notin K$.

   a. Geef een deterministische eindige automaat voor K.

   b. Toon aan dat de taal K regulier is door deze uit te drukken in eindige talen met behulp van de operaties vereniging, concatenatie en ster.

   De reguliere taal L is als volgt gedefinieerd:

$$L = \{a, b\} \cdot \{a\} \cdot \{a, b\}^* \cdot \{a\} \cdot \{a, b\} \cup \{a, b\} \cdot \{b\} \cdot \{a, b\}^* \cdot \{b\} \cdot \{a, b\}$$

   c. Beschrijf L in woorden, analoog aan de manier waarop in onderdeel a. de taal K gedefinieerd is

3. a. Geef een deterministische eindige automaat voor de taal

$$K = \{\ w \in \{a, b\}^* \mid w \text{ heeft een suffix } abb\ \}$$

   b. Doe dit ook voor de taal $\text{mir}(K)$. (spiegelbeeld)

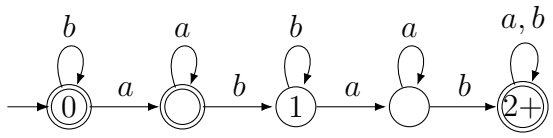
   c. Toon aan dat K *regulier* is, maw. druk K uit in eindige talen met behulp van de operaties vereniging, concatenatie en ster ($\cup$, $\cdot$, *).

   Doe dit ook voor het *complement* van K ten opzichte van $\{a, b\}^*$.

1) Alle woorden met één voorkomen *ab*.

a. Zowel *aba* als *bab* behoren tot K (want bevatten elk één voorkomen van *ab*). Dan behoort $aba \cdot bab$ tot K^2, maar dit woord heeft niet twee, maar drie voorkomens van *ab*.

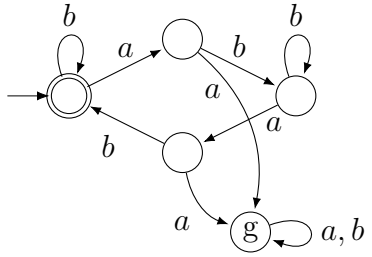
b. Complement: geen *ab* of tenminste twee.



c. $\{b\}^*\{a\}^* \cup \{a,b\}^*\{ab\}\{a,b\}^*\{ab\}\{a,b\}^*$,
of als reguliere expressie $b^*a^* + (a+b)^*ab(a+b)^*ab(a+b)^*$.

2) Even aantal a, elke a door een b gevolgd.

a.

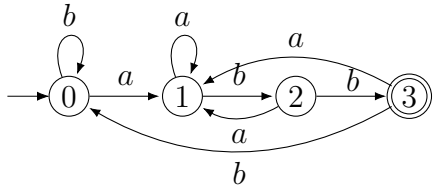


b. $(\ \{b\}^*\{ab\}\{b\}^*\{ab\}\ )^*\{b\}^*$.

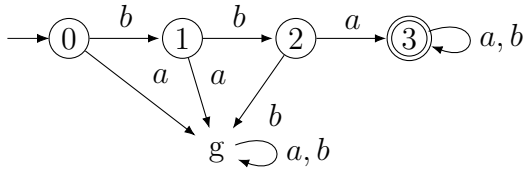
c. Tweede letter gelijk aan de een-na-laatste letter (en lengte tenminste 4).

3) Suffix *abb*.

a.



b. Spiegelbeeld: prefix *bba*.



c. $\{a,b\}^*\{abb\}$

Complement, eindigt niet op *abb*, dus eindigt op a, op *ab*, of op *bbb*. Maar we zijn de korte woorden vergeten! $\{\lambda, b, bb\} \cup \{a,b\}^*\{a, ab, bbb\}$

M 1.33. Let L_1 and L_2 be subsets of $\{a, b\}^*$.

a. Show that if $L_1 \subseteq L_2$, then $L_1^* \subseteq L_2^*$.

b. Show that $L_1^* \cup L_2^* \subseteq (L_1 \cup L_2)^*$.

c. Give an example of two languages L_1 and L_2 such that $L_1^* \cup L_2^* \neq (L_1 \cup L_2)^*$.

d. One way for the two languages $L_1^* \cup L_2^*$ and $(L_1 \cup L_2)^*$ to be equal is for one of the two languages L_1 and L_2 to be a subset of the other, or more generally, for one of the two languages L_1^* and L_2^* to be a subset of the other. Find an example of languages L_1 and L_2 for which neither of L_1^* and L_2^*, but $L_1^* \cup L_2^* = (L_1 \cup L_2)^*$.

M 1.36.

a. Consider the language L of all strings of a's and b's that do not end with b and do not contain the substring bb. Find a finite language S such that $L = S^*$.

b. Show that there is no language S such that S^* is the language of all strings of a's and b's that do not contain the substring bb.

M 1.37. Let L_1, L_2, and L_3 be languages over some alphabet Σ. In each case below, two languages are given. Say what the relationship is between them. (Are they always equal? If not, is one always a subset of the other?) Give reasons for your answers, including counterexamples if appropriate.

a. $L_1(L_2 \cap L_3)$, $\quad L_1L_2 \cap L_1L_3$

b. $L_1^* \cap L_2^*$, $\quad (L_1 \cap L_2)^*$

c. $L_1^*L_2^*$, $\quad (L_1L_2)^*$

Opgaven aangegeven met **M** komen uit het boek van John C. Martin, *Introduction to Languages and the Theory of Computation*, 4th edition, McGraw Hill, 2010

De uitwerkingen van deze opgaven zijn grotendeels gebaseerd op eerdere versies van FI2, door collega's J. Kleijn en M.M. Bonsangue.

1.33 Let $L_1, L_2 \subseteq \{a,b\}^*$.

a. Assume $L_1 \subseteq L_2$. Then $LL_1 \subseteq LL_2$, for any language L: $LL_1 = \{uv \mid u \in L, v \in L_1\} \subseteq \{uv \mid u \in L, v \in L_2\} = LL_2$. Symmetrically $L_1L \subseteq L_2L$.

Thus $L_1^2 = L_1L_1 \subseteq L_1L_2 \subseteq L_2L_2L_2^2$ and in general $L_1^k = L_1^{k-1}L_1 \subseteq L_1^{k-1}L_2 \subseteq L_2^{k-1}L_2 = L_2^k$ for all $k \geq 1$. Consequently,

$L_1^* = \bigcup_{k\geq 0} L_1^k \subseteq \bigcup_{k\geq 0} L_2^k = L_2^*$.

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 $L_1 \subseteq L_1^* \subseteq L_2^*$ also $(L_1 \cup L_2)^* \subseteq (L_2^* \cup L_2)^* = L_2^*$. Similarly, $L_2^* \subseteq L_1^*$ implies that $L_1^* \cup L_2^* = L_1^* = (L_1 \cup L_2)^*$.

Next consider $L_1 = \{0^2, 0^5\}$ 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 $S^* = S^*S^*$, it follows that $bb \in S^*S^* = S^* = K$, 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 Σ.

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 $w \in L_1L_3$.

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

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 $L_1^* \cap L_2^* = \{a\}^* \cap \{aa\}^* = \{aa\}^* \neq \{\lambda\} = \varnothing^* = (L_1 \cap L_2)^*$.

c. $L_1^*L_2^*$ 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 $abab \in (L_1L_2)^* - L_1^*L_2^*$.

Als extra oefening.

M 2.1. Draw a DFA accepting for each of the following languages, The alphabet is $\{a, b\}$. The language of all strings ...

a. ... containing exactly two a's.

b. ... containing at least two a's.

c. ... that do not end with ab.

d. ... that begin or end with aa or bb.
(i) ... that begin with aa or bb. (ii) ... that end with aa or bb.

e. ... not containing the substring aa.

f. ... in which the number of a's is even.

g. ... in which both the number of a's and the number of b's are even.

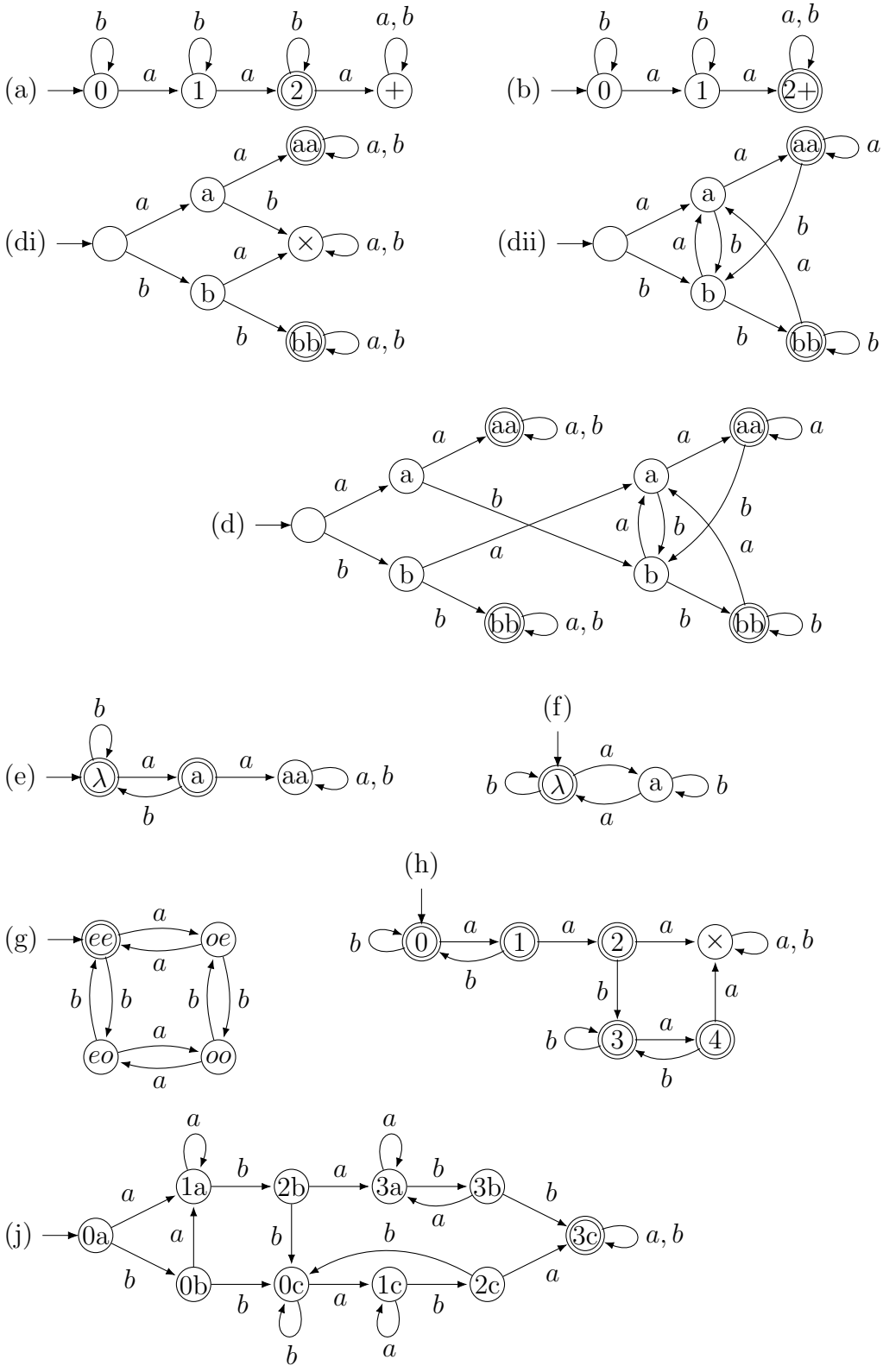
h. ... containing no more than one occurrence of the string aa.
(The string aaa contains two occurrences of aa.)

i. ... in which every a (if there are any) is followed immediately by bb.

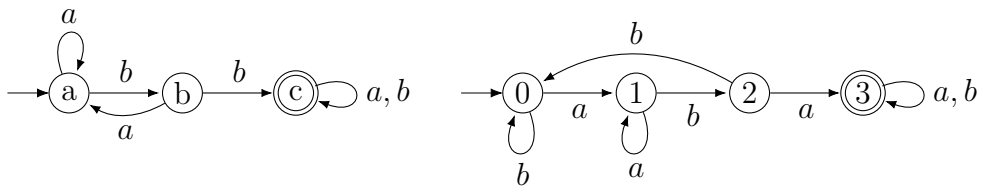
j. ... containing both bb and aba as substrings.

k. ... containing both aba and bab as substrings.

M 2.1

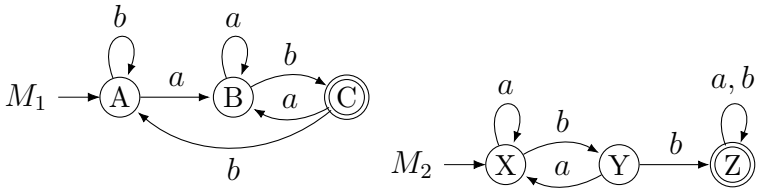


This DFA can be found 'manually', but also by the *product construction* for the intersection of the languages for substring *bb* and *aba* (a parallel simulation of both automata).


a
b
c
a
b
b
a, b
0
1
2
3
a
b
a
b
a, b

M 2.10. De twee automaten M_1 en M_2 hieronder accepteren respectievelijk de talen L_1 en L_2.
Construeer DFA voor elk van de volgende talen.

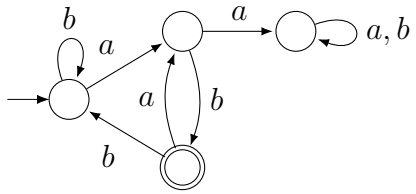
a. $L_1 \cup L_2$ b. $L_1 \cap L_2$ c. $L_1 - L_2$



M 2.12. For each of the following languages, draw a DFA accepting it.

a. $\{a,b\}^*\{a\}$
b. $\{bb, ba\}^*$
c. $\{a,b\}^*\{b, aa\}\{a,b\}^*$
d. $\{bbb, baa\}^*\{a\}$
e. $\{a\} \cup \{b\}\{a\}^* \cup \{a\}\{b\}^*\{a\}$
f. $\{a,b\}^*\{ab, bba\}$
g. $\{b, bba\}^*\{a\}$
h. $\{aba, aa\}^*\{ba\}^*$

M 2.13. For the DFA pictured below, show that there cannot be any DFA with fewer states accepting the same language.



M 2.14. Let z be a fixed string of length n over the alphabet $\{a,b\}$.
We can find a DFA with $n+1$ states accepting the language of all strings in $\{a,b\}^*$ that end in z. Its states correspond to the $n+1$ distinct prefixes of z.
Show that there can be no DFA with fewer than $n+1$ states accepting this language.

M 2.21. For each of the following languages $L \subseteq \{a,b\}^*$, show that the elements of the infinite set $\{\ a^n \mid n \geq 0\ \}$ are pairwise L-distinguishable.

a. $L = \{\ a^n b a^{2n} \mid n \geq 0\ \}$
b. $L = \{\ a^i b^j a^k \mid k > i + j\ \}$
c. $L = \{\ a^i b^j \mid j = i \text{ or } j = 2i\ \}$
d. $L = \{\ a^i b^j \mid j \text{ is a multiple of } i\ \}$
e. $L = \{\ x \mid n_a(x) < 2n_b(x)\ \}$
f. $L = \{\ x \mid$ no prefix of x has more $b's$ than $a's\ \}$
g. $L = \{a^{n^3} \mid n \geq 1\ \}$
h. $L = \{\ ww \mid w \in \{a,b\}^*\ \}$

We conclude that none of these language is accepted by a DFA.

M 2.10 Apply the product construction to M_1 and M_2:

	a	b
(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)

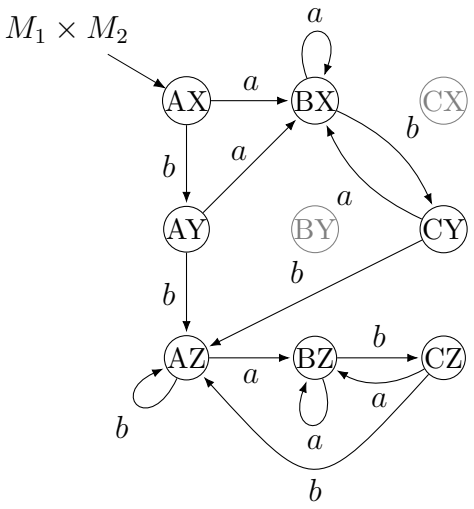
The desired automata each have (A, X) as initial state. Note that the states (B, Y) and (C, X) which are not reachable from (A, X) are not mentioned in the table for the product transition function.

a. 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.

b. 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.

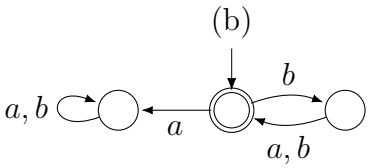
c. For $L_1 - L_2$, we have as accepting states $\{(C, Y)\}$, that is any pair of original states in which the first is accepting and the second not.

Drawing the automata is now easy; add the proper accepting states for each application.

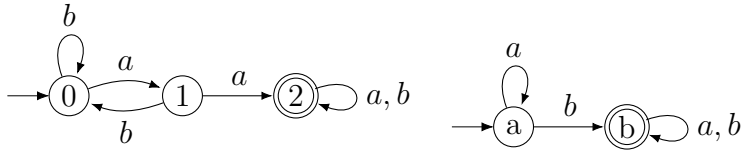


M 2.12 In het volgende hoofdstuk leren we om op een systematische manier reguliere operaties te vertalen naar eindige automaten. Het is me niet altijd duidelijk welke methode het boek in gedachten heeft.

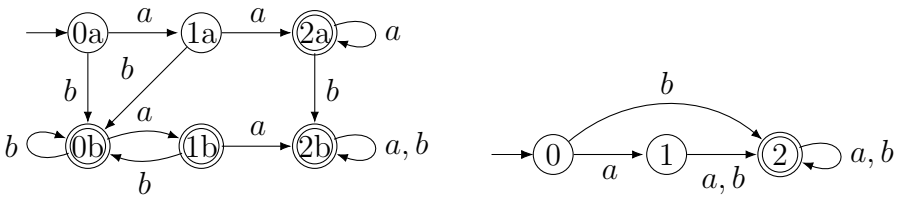
b. $\{bb, ba\}^*$: strings of even length, every odd position is a b (counting the first position as 1).



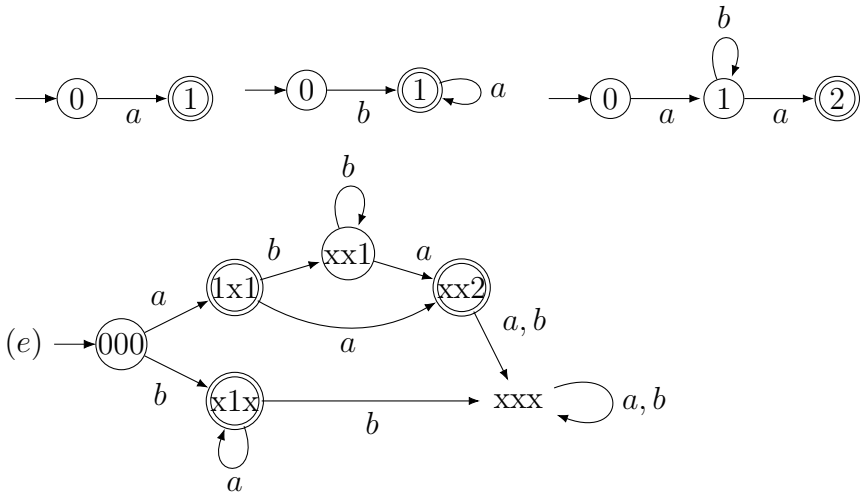
c. Can be written as the union of $\{a, b\}^*\{aa\}\{a, b\}^*$ (contains substring aa) and $\{a, b\}^*\{b\}\{a, b\}^*$ (contains b).



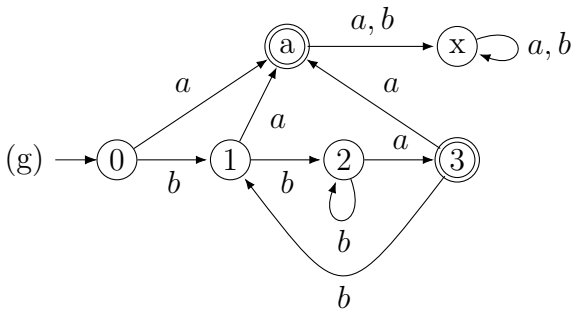
Then we can apply the product construction, for union. Obviously, this is not the most compact automaton. Observe that only two strings are not in the language.



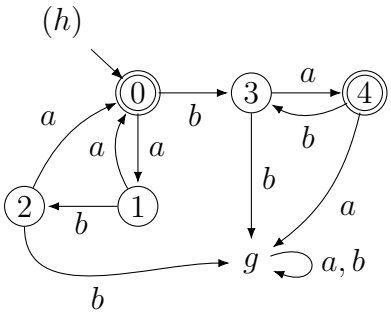
e. The union of languages indicates three easily handled cases. For each automaton we omitted the garbage (or sink) state x. Simulate in parallel.



g. Verbeterd! Elke a is óf de laatste a van de string, of wordt vooraf gegaan door bb. Verder moet de laatste letter een a zijn. We tellen dus de b's om te kijken of de volgende a aan de eis voldoet. In toestand 3 hebben we een a gezien die 'aan de regels' voldoet. Dat mag de laatste zijn (want volgt dan gewoon op twee losse b's, maar er mag ook een a op volgen (en maakt dan deel uit van bba).



h. Ook hier geen idee hoe ik hier een productconstrutie kan toepassen?



M 2.13 The automaton accepts the language L consisting of all strings from $\{a, b\}^*$ that do not contain aa and end in ab.

The simplest strings corresponding to its four states are λ, a, ab, and aa. If each pair of these strings are distinguishable, then it follows from Theorem 2.21 that any DFA recognizing L has at least 4 states. Strings x, y are distinguishable if we find z such that exactly one of xz and yz belong to L.

First note that an accepting state is distinguishable with any nonaccepting state. In this case, ab versus λ, a, aa. Choose $z = \lambda$: $ab \in L$, while a, b and aa are not in L. In the table below we choose some z such that every pair of the four strings is shown to be distinguishable.

$x =$	λ	a	ab	aa
$z = \lambda$	$\lambda \notin L$	$a \notin L$	$ab \in L$	$aa \notin L$
ab	$ab \in L$	$aab \notin L$	$abab \in L$	$aaab \notin L$
b	$b \notin L$	$ab \in L$	$abb \notin L$	$aab \notin L$

Hence, there is no DFA accepting L with fewer states than the given DFA.

M 2.14 Let z be a word over the alphabet $\{a, b\}$. $L = \{a, b\}^*\{z\}$ is the language of all strings that end in z (have suffix z). Any DFA which accepts L has at least $|z| + 1$ states. The reason is that z has $|z| + 1$ prefixes (from λ to z itself) which all have to be distinguished in the automaton. Consider two prefixes x and u of z, with $|x| > |u|$, and let $z = xy$. Then $xy = z \in L$, but $uy \notin L$ (t is shorter than z, so cannot end in z). 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. We will formally prove this.

Consider a word x and let v be its longest suffix which is also a prefix of z: thus $x = uv$ and $z = vw$ for some w. (Note that *every* word x has such a suffix!)

We will show that $x \equiv_L v$, or, for every $y \in \{0, 1\}^*$ we have $xy \in L$ iff $vy \in L$. Or, equivalently, xy ends in z iff vy ends in z.

Choose any y. We consider two cases.

(1) $|y| \geq |z|$, as z is shorter than y, we have $xy \in L$ iff $y \in L$ iff $vy \in L$ because it is only relevant whether y ends with z or not;

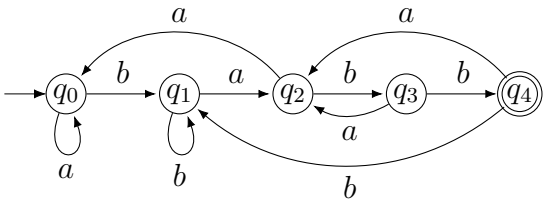
(2) case $|y| < |z|$. Then xy ends in z iff there is v' such that $x = u'v'$ with $v'y = z$. Thus, v' is a suffix of x that is also a prefix of z. Recall that v is the longest suffix of x with this property: thus $|v| \geq |v'|$ and v' is a suffix of v. That means $z = v'y$ is a suffix of vy.

(Dit zou twee kanten op moeten gaan, deze implicatie.)

Consequently, x and v are indistinguishable and we know how to define a DFA accepting L.

As an example we give a DFA accepting $\{a, b\}^*\{babb\}$ with 5 states:

q_0 corresponding to matching suffix λ; 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 $z = babb$.



M 2.21 In each case we consider a^i and a^ℓ such that $i \neq \ell$, and we find string z such that exactly one of a^iz and $a^\ell z$ belongs to L. This shows that each pair of strings in $\{a\}^*$ is L-distinguishable.

a. Let $z = ba^{2i}$. Then $a^iz \in L$, while $a^\ell z \notin L$

b. Assume that $\ell > i$. Let $z = ba^{i+2}$.

c. Again assume that $\ell > i$. Let $z = b^i$. Then $a^ib^i \in L$, while $a^\ell b^i \notin L$, because $i < \ell$, so neither $i = \ell$ nor $i = 2\ell$.

f. Assume that $\ell > i$. Choose $z = b^\ell$. Then $a^ib^\ell \notin L$, while $a^\ell b^\ell \in L$.

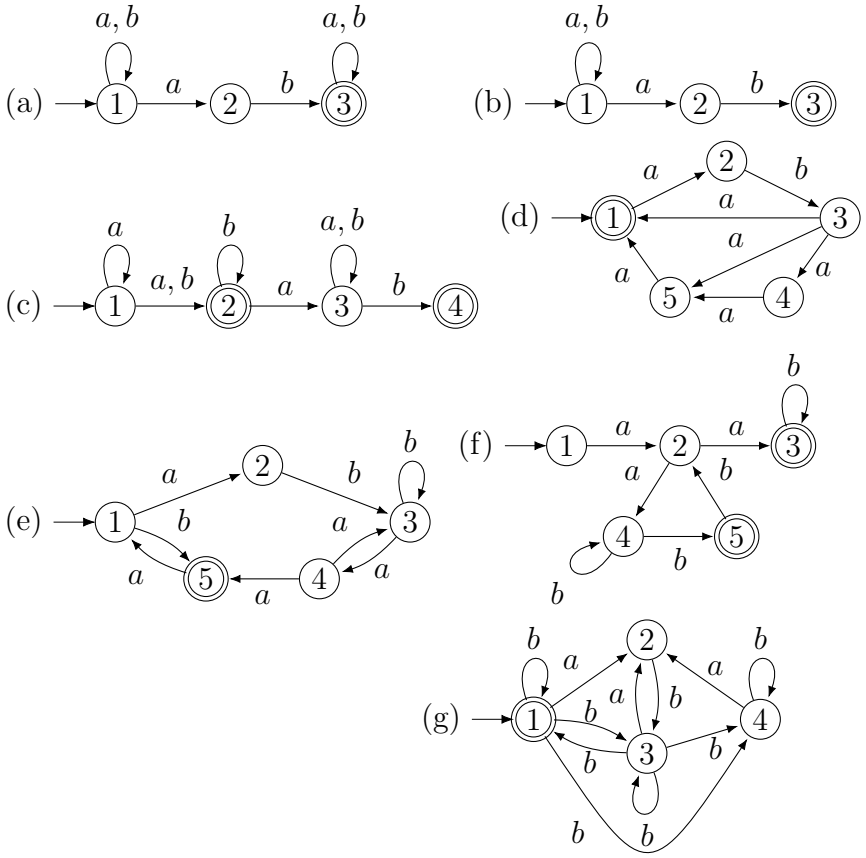
g. It is easy to choose $z = n^3 - i$ for some value $n^3 \geq i$, so that $a^iz = a^{n^3} \in L$, but we have to argue that $a^\ell z \notin L$. (An example where the argument would go wrong is a^{20} vs. a^{57}. Choosing $z = a^7$ we have two cubes: $20 + 7 = 27$ and $57 + 7 = 64$.)

Again assume $\ell > i$. Use the fact that consecutive cubes n^3 are at increasing distances. Choose n such that the distance $(n + 1)^3 - n^3$ is larger than $\ell - i$. Then $\ell + n^3 - i$ is in between n^3 and $(n + 1)^3$.

X.1. For each of the following languages, draw a NFA accepting it. Compare with M 2.12.

a.	$\{a,b\}^*\{a\}$	e.	$\{a\} \cup \{b\}\{a\}^* \cup \{a\}\{b\}^*\{a\}$
b.	$\{bb, ba\}^*$	f.	$\{a,b\}^*\{ab, bba\}$
c.	$\{a,b\}^*\{b, aa\}\{a,b\}^*$	g.	$\{b, bba\}^*\{a\}$
d.	$\{bbb, baa\}^*\{a\}$	h.	$\{aba, aa\}^*\{ba\}^*$

3.38. Below we have depicted several NFA. Using the subset construction, for each of them construct a DFA accepting the same language. Indicate the subsets in the states.

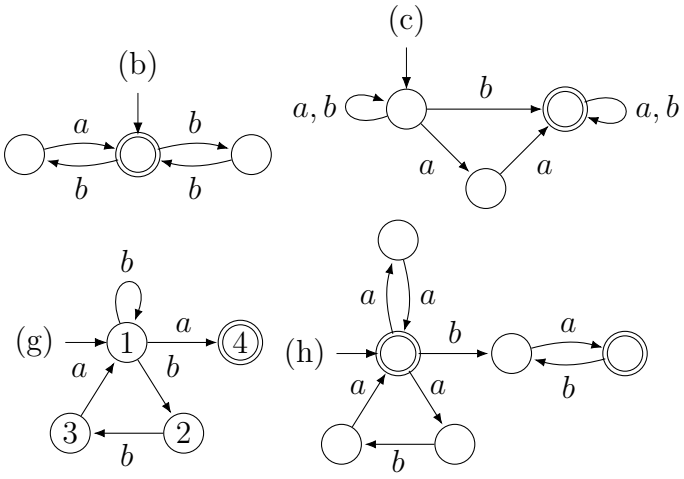


3.44. Suppose $M = (Q, \Sigma, \delta, q_{in}, A)$ is an NFA accepting a language L.

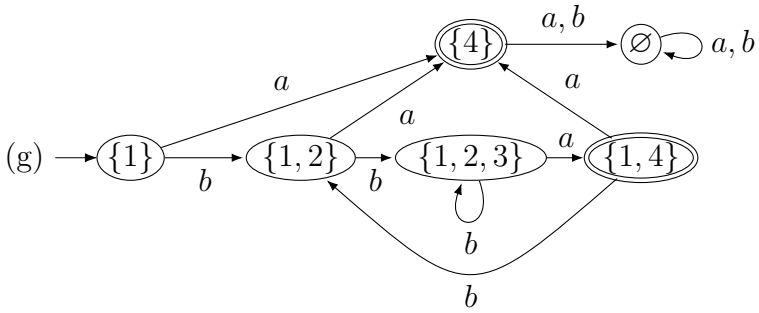
a. Describe how to construct an NFA M_1 with no transitions to its initial state so that M_1 also accepts L.

b. Investigate how to construct an NFA M_2 with exactly one accepting state and no transitions from that state, so that M_2 also accepts L.

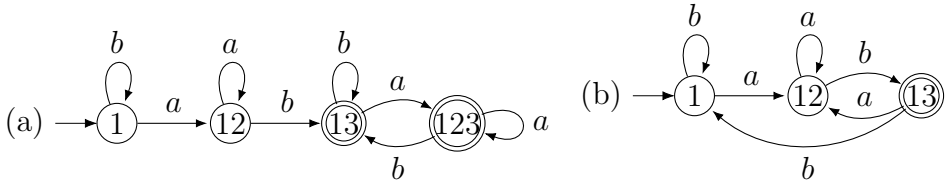
X.1.

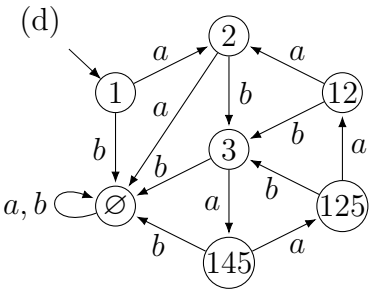


Laten we (g) eens deterministisch maken. We krijgen de automaat de we eerder al 'met de hand' gemaakt hebben.

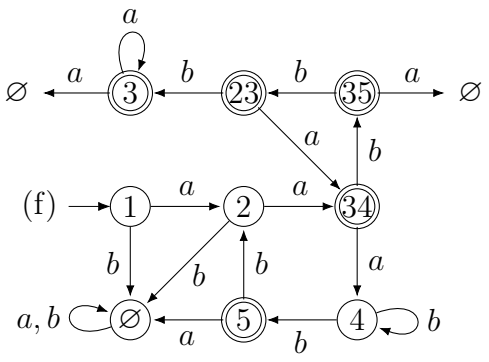


3.38. In de oplossingen stelt bijvoorbeeld 123 natuurlijk de verzameling $\{1, 2, 3\}$ voor.





Om geklodder te vermijden hebben we hieronder de garbage toestand $\varnothing$ een aantal keer getekend.



3.44 a. Let $M = (Q, \Sigma, \delta, q_{in}, A)$ be an NFA.

We add a new initial state $q'_{in} \notin Q$ and copy all the outgoing edges from the original q_{in} to q_{in}: $(q'_{in}, \sigma, q) \in \delta'$ whenever $(q_{in}, \sigma, q) \in \delta$. New q'_{in} will be accepting when $q_i ni$ is accepting.

All the rest remains unchanged: $M' = (Q \cup \{q'_{in}\}, \Sigma, \delta', q'_{in}, A')$.

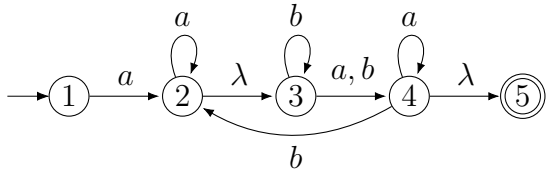
b. We difficulties when the initial state is also accepting. Then in the new automaton that must also be the case, as this is the only way to accept the empty string. If the initial state is the only accepting state, we will always accpet the Kleene star of all accepted strings.

Otherwise, if the initial state is not accepting, we can mimic the previous construction. Let $M = (Q, \Sigma, \delta, q_{in}, A)$ be an NFA.

We add a new accepting state $q_f \notin Q$ and copy all edges incoming to any accepting state so to also go to q_f: $(p, \sigma, q_f) \in \delta'$ whenever $(p, \sigma, q) \in \delta$ for some $q \in A$.

All the rest remains unchanged: $M' = (Q \cup \{q_f\}, \Sigma, \delta', q_{in}, \{q_f\})$.

M 3.18. Below the transition diagram for a NFA-λ. For each string below, say whether the automaton accepts it. **a.** *aba* **b.** *abab* **c.** *aaabbb*

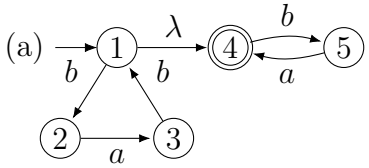


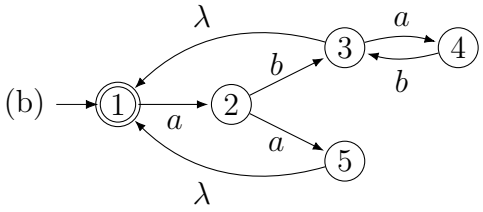
M 3.22. A transition table is given for a NFA-λ with seven states.

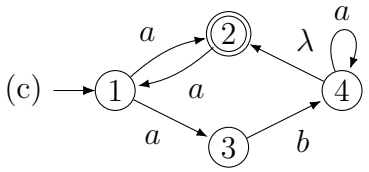
q	$\delta(q,a)$	$\delta(q,b)$	$\delta(q,\lambda)$
1	$\varnothing$	$\varnothing$	$\{2\}$
2	$\{3\}$	$\varnothing$	$\{5\}$
3	$\varnothing$	$\{4\}$	$\varnothing$
4	$\{4\}$	$\varnothing$	$\{1\}$
5	$\varnothing$	$\{6,7\}$	$\varnothing$
6	$\{5\}$	$\varnothing$	$\varnothing$
7	$\varnothing$	$\varnothing$	$\{1\}$

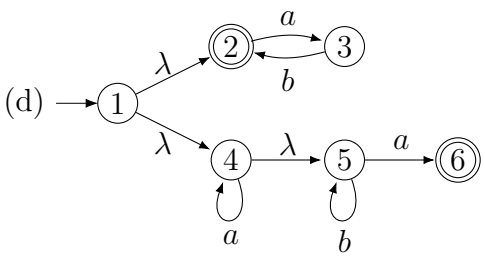
Find: **a.** $\Lambda(\{2,3\}$ **b.** $\Lambda(\{1\}$ **c.** $\Lambda(\{3,4\}$
d. $\delta^*(1,ba)$ **e.** $\delta^*(1,ab)$ **f.** $\delta^*(1,ababa)$

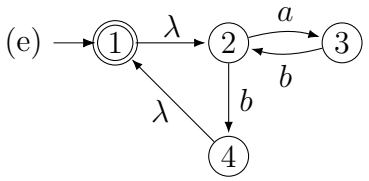
M 3.37. Draw equivalent NFA (so, without λ-transitions).



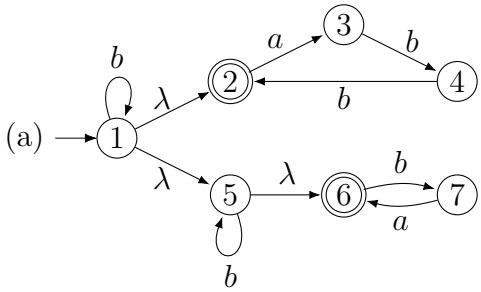


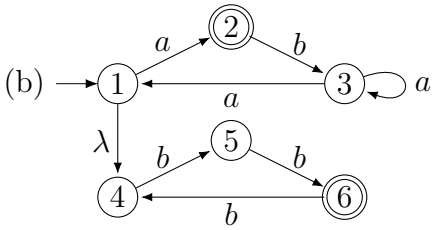






M 3.40. Draw equivalent DFA.





M 3.18 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 $\delta^*(1, abab) = \Lambda(\delta(4, b) \cup \delta(5, b)) = \Lambda(\{2\}) = \{2, 3\}$.

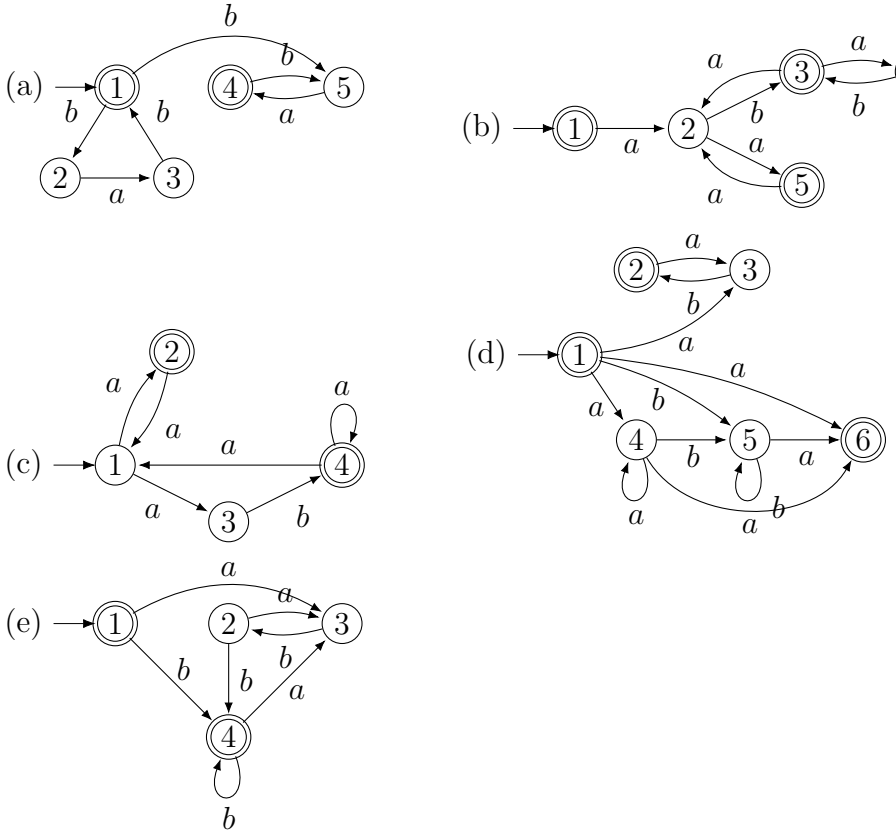
c. $aaabbb$ is accepted by M (check!).

M 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 $\delta^*(1, ba) = \Lambda(\bigcup_{p \in \delta^*(1,b)} \delta(p, a)) = \Lambda(\delta(1, a) \cup \delta(2, a) \cup \delta(5, a) \cup \delta(6, a) \cup \delta(7, a)) = \Lambda(\{3, 5\}) = \{3, 5\}$.

M 3.37. Merk op dat (a) een voor de hand liggende representatie is van de taal $\{bab\}^*\{ba\}^*$. Evenzo is (d) een automaat voor $\{ab\}^* \cup \{a\}^*\{b\}^*\{a\}$.



M 3.1. In each case below, find a string of minimum length in $\{a, b\}^*$ not in the language corresponding to the given regular expression.

a. $b^*(ab)^*a^*$

b. $(a^* + b^*)(a^* + b^*)(a^* + b^*)$

c. $a^*(baa^*)^*b^*$

d. $b^*(a + ba)^*b^*$

M 3.3. Let r and s be arbitrary regular expressions over the alphabet Σ.
In each case below, find a simpler equivalent regular expression.

a. $r(r^*r + r^*) + r^*$

b. $(r + \Lambda)^*$

c. $(r + s)^*rs(r + s)^* + s^*r^*$

M 3.6. Suppose w and z are strings in $\{a, b\}^*$. Find regular expressions corresponding to each of the languages defined recursively below.

a. $\lambda \in L$; for every $x \in L$, then wx and xz are elements of L.

b. $a \in L$; for every $x \in L$, wx, xw, and xz are elements of L.

c. $\lambda \in L$; $a \in L$; for every $x \in L$, wx and zx are in L.

M 3.7. Find a regular expression corresponding to each of the following subsets of $\{a, b\}^*$.
The language of all strings ...

a. ... containing exactly two a's.

b. ... containing at least two a's.

c. ... that do not end with ab.

d. ... that begin or end with aa or bb.

e. ... not containing the substring aa.

f. ... in which the number of a's is even.

g. ... containing no more than one occurrence of the string aa.
(aaa contains two occurrences of aa.)

h. ... in which every a is followed immediately by bb.

i. ... containing both bb and aba as substrings.

j. ... not containing the substring aaa.

k. ... not containing the substring bba.

l. ... containing both bab and aba as substrings.

m. ... in which the number of a's is even and the number of b's is odd.

n. ... in which both the number of a's and the number of b's are odd.

M 3.10. a. If L is the language corresponding to the regular expression $(aab{+}bbaba)^*baba$, find a regular expression corresponding to $L^r = \{ x^r \mid x \in L \}$.

b. Using the example in part (a) as a model, give a recursive definition of the reverse e^r of a regular expression e.

c. Show that for every regular expression e, if the language L corresponds to e, then L^r corresponds to e^r.

M 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 Λ, 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 ≤ 3 belong to the language.

M 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 $(r + s)^*$.

M 3.6 a. $(w)^*(z)^*$.

b. $(w)^*a(w + z)^*$.

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

M 3.7 c. Empty string, end in a or end in bb: $(a + b)^*(a + bb) + \Lambda$.

e. Every a is followed by a b, unless it is the last letter: $(b + ab)^*(\Lambda + a)$.

k. Once we have seen bb only b's can follow: $(a + ba)^*b^*$.

l. $(a + b)^*bab(a + b)^*aba(a + b)^* + (a + b)^*aba(a + b)^*bab(a + b)^* + (a + b)^*baba(a + b)^* + (a + b)^*abab(a + b)^*$.

m. $(a + b(aa + bb)^*(ab + ba))(aa + bb + (ab + ba)(aa + bb)^*(ab + ba))^*$

The first part checks a minimum string with an odd number of b's and even number of a's. After that the strings can be extended as much as we want with a string of even number of a's and b's .

n. $(aa+bb+(ab+ba)(aa+bb)^*(ab+ba))^*(ab+ba)(aa+bb+(ab+ba)(aa+bb)^*(ab+ba))^*$.

The central part denotes the string with one a and one b. And then we add left and right an even number of a's and b's.

M 3.10 The reverse function `rev` assigns to each string its reversal (mirror image).

Formally, given an alphabet Σ, we define `rev`: $\Sigma^* \to \Sigma^*$ recursively by:

`rev`$(\lambda) = \lambda$ (no change)

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

Now `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):

$\texttt{rrev}(\varnothing) = \varnothing$; $\texttt{rrev}(\Lambda) = \Lambda$; $\texttt{rrev}(a) = a$ for all $a \in \Sigma$.

and for the composite elements:

if e_1 and e_2 are regular expressions, then

– $\texttt{rrev}(e_1 + e_2) = \texttt{rrev}(e_1) + \texttt{rrev}(e_2)$;
– $\texttt{rrev}(e_1 e_2) = \texttt{rrev}(e_2)\,\texttt{rrev}(e_1)$; and
– $\texttt{rrev}(e_1^*) = \texttt{rrev}(e_1))^*$.

Now we have to prove that this `rrev` has the property $\|\,\texttt{rrev}(e)\| = \|e\|^r$.

This is proved by induction on the structure of e.

Basis first.

– $e = \varnothing$: then $\|\,\texttt{rrev}(\varnothing)\| = \|\varnothing\| = \varnothing = \|\varnothing\|^r$;
– $e = \Lambda$: then $\|\,\texttt{rrev}(\Lambda)\| = \|\Lambda\| = \{\lambda\} = \|\Lambda\|^r$;
– $e = a$: then $\|\,\texttt{rrev}(a)\| = \|a|\ = \{a\} = \|a\|^r$.

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

– $e = e_1 + e_2$: then
$\|\,\texttt{rrev}(e_1 + e_2)\| = \|\,\texttt{rrev}(e_1) + \texttt{rrev}(e_2)\| = \|\,\texttt{rrev}(e_1)\| \cup \|\,\texttt{rrev}(e_2)\| =$
(induction) $\|(e_1)\|^r \cup \|(e_2)\|^r = (\|e_1\| \cup \|e_2\|)^r = \|e_1 + e_2\|^r$;

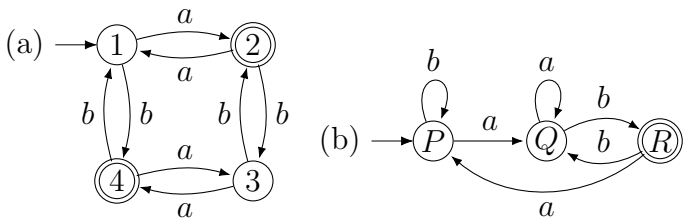
– $e = e_1 e_2$: then
$\|\,\texttt{rrev}(e_1 e_2)\| = |\,\texttt{rrev}(e_2)\,\texttt{rrev}(e_1)\| = \|\,\texttt{rrev}(e_2)\| \cdot \|\,\texttt{rrev}(e_1)\| =$
(induction) $\|(e_2)\|^r \|(e_1)\|^r = (\|(e_1)\| \cdot \|(e_2)\|)^r = \|e_1 e_2\|^r$;

– $e = e_1^*$: then
$\|\,\texttt{rrev}(e_1^*)\| = \|(\texttt{rrev}(e_1))^*\| = \|\,\texttt{rrev}(e_1)\|^* =$
(induction) $(\|e_1\|^r)^* = (\|e_1\|^*)^r = \|e_1^*\|^r$.

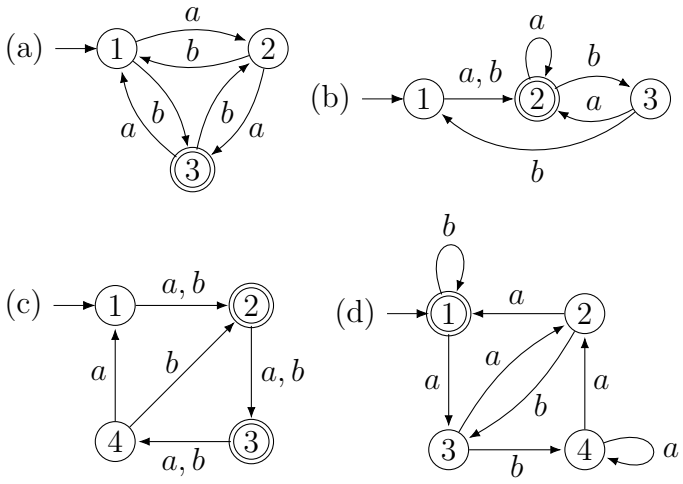
c. It follows from **b.** that the language L^r is regular whenever the language L is regular: we have seen that $L = \|e\|$ implies that $L^r = \|\,\texttt{rrev}(e)\|$ and that $\texttt{rrev}(e)$ is a regular expression follows immediately from the definition of `rrev` as given above.

M 3.49. Below two DFA M_1 and M_2 accepting languages L_1 and L_2, respectively respectively. Draw NFA-λ accepting each of the following languages, using the constructions for the regular operations.

a. $L_2^* \cup L_1$ **b.** $L_2 L_1^*$ **c.** $L_1 L_2 \cup (L_2 L_1)^*$



M 3.51. Bepaal een reguliere expressie voor elk van de onderstaande DFA, met behulp van een geschikt algoritme.



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 (a) as a system of 3 equations in three variables:

$$\begin{aligned} x_1 &= ax_3 + bx_2 \\ x_2 &= ax_1 + bx_3 \\ x_3 &= ax_2 + bx_1 + \Lambda \end{aligned}$$

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

$$\begin{aligned} x_1 &= a(ax_2 + bx_1 + \Lambda) + bx_2 = abx_1 + (aa + b)x_2 + a \\ x_2 &= ax_1 + b(ax_2 + bx_1 + \Lambda) = bax_2 + ((a + bb)x_1 + b) \end{aligned}$$

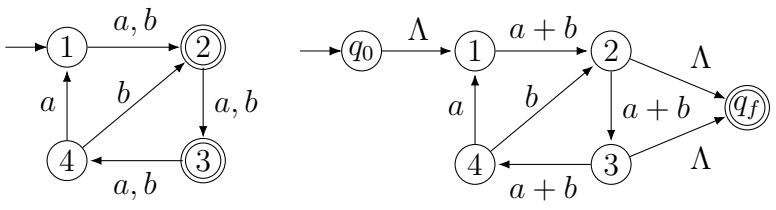
Using the Arden's lemma, we obtain that $x_2 = (ba)^*((a + bb)x_1 + b)$. If we substitute x_2 in the first equations we have

$$\begin{aligned} x_1 &= abx_1 + (aa + b)(ba)^*((a + bb)x_1 + b) + a \\ &= (ab + (aa + b)(ba)^*(a + bb))x_1 + ((aa + b)(ba)^*b + a) \end{aligned}$$

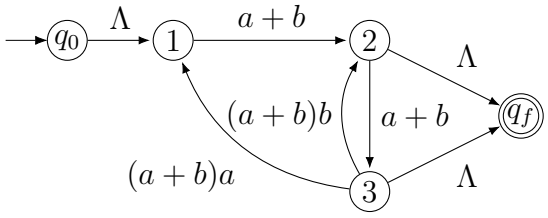
Using again Arden's lemma, we obtain that $x_1 = ((ab + (aa + b)(ba)^*(a + bb)))^*((aa + b)(ba)^*b + a)$. This is a regular expression denoting the same language of the automaton in Figure 3.40 (a).

3.51 c We use the *state removal method* of Brzozowski and McCluskey to derive for the depicted automata a corresponding regular expression.

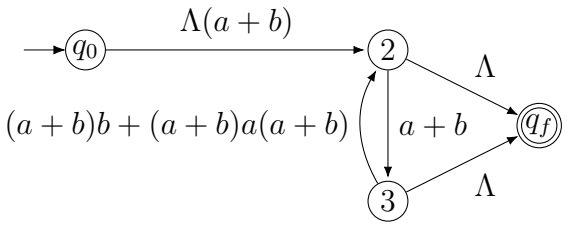
First we add a new initial state q_0 without incoming transitions and a new (the only) final state q_f without outgoing transitions in such a way that the resulting NFA accepts the same language (see exercise 3.44). At the same time we combine with + the labels of parallel edges into a single regular expression.



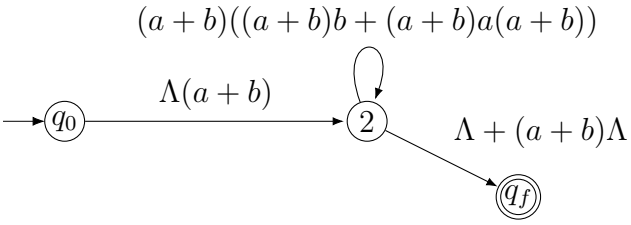
Now we remove state 4. Before deleting 4, we consider the transitions from 3 to 4 and from 4 to 1 and 2. This leads to the introduction of an arc labeled with $(a + b)a$ from 3 to 1 and an arc labeled with $(a + b)b$ from 3 to 2.



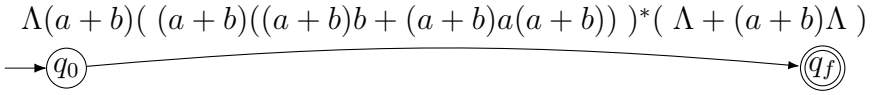
Then state 1 is removed. This leads to the introduction of an arc labeled with $\Lambda(a + b)$ from q_0 to 2 and an arc labeled with $(a + b)a(a + b)$ from 3 to 2. The latter is combined using + with the label $(a + b)b$ of the already existing arc from 3 to 2.



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 Λ. 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 $L(M)$.



In set notation we have $L(M) = \{a,b\}(\{a,b\}(\{a,b\}\{b\} \cup \{a,b\}\{a\}\{a,b\}))^*\{\Lambda,a,b\}$.

M 2.22. Gebruik voor elk van de talen uit opgave M 2.21 (bladzijde 8) het pomplemma om te laten zien dat de taal niet door een DFA geaccepteerd kan worden.

M 2.57. Each case below defines a language over $\{a, b\}$. Determine whether these languages can be accepted by a DFA, and prove that your answer is correct. The set of all strings ...

a. ... x beginning with a nonnull[=non-empty] string of the form ww.

b. ... x containing some nonnull substring of the form ww.

c. ... x having some nonnull substring of the form www. (use the following fact: there are arbitrarily long strings in $\{a, b\}^*$ that do not contain any nonnull substring of the form www.)

d. ... of odd-length with middle symbol a.

e. ... of even-length, and of length at least 2 with the two middle symbols equal.

f. ... of the form xyx for some x with $|x| \geq 1$.

g. The set of non-palindromes.

h. ... in which the number of a's is a perfect square.

i. ... having the property that in every prefix, the number of a's and the number of b's differ by no more than 2.

j. ... having the property that in some prefix, the number of a's is 3 more than the number of b's.

k. ... in which the number of a's and the number of b's are both divisible by 5.

l. ... x for which there is an integer $k > 1$ (possibly depending on x) such that the number of a's in x and the number of b's in x are both divisible by k.

Assuming that L can be accepted by a DFA,

m. $\max(L) = \{x \in L \mid$ there is no nonnull string y so that $xy \in L\}$.

n. $\min(L) = \{x \in L \mid$ no prefix of x other than x itself is in $L\}$.

2.57 You are asked for a number of languages over the alphabet $\{a, b\}$ to determine whether they can be accepted by a DFA or not. You can go after the number of equivalence classes, or use the pumping lemma for a negative answer. To show the language is regular you can build a DFA or use closure properties (like Boolean operations) on known languages.

a. No. Let $L = \{x \in \{a,b\}^* \mid \exists w, y \in \{a,b\}^*.w \neq \Lambda \wedge x = wwy\}$, 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. Yes. Let $L = \{x \in \{a,b\}^* \mid \exists y, w, z : w \neq \Lambda \wedge x = ywwz\}$.

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

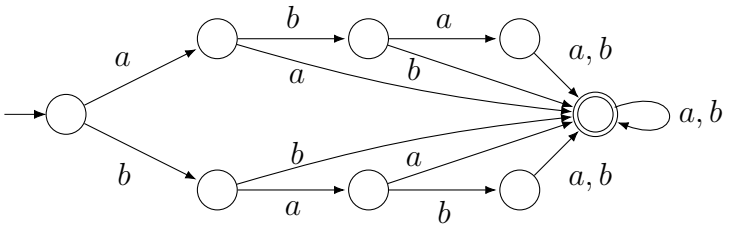
Proof of the claim: if aa, bb, $abab$ of $baba$ a subword is of x, then is x by definition in L.

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 $x = ywwz$. 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 $w = ababv$ or $w = babav$, respectively, en we are ready also in this case.

The only case that remains to check is when $|u| = 1$. If $w = abu$, than must be $u = a$ and $x = ywwz = yabaabaz$, contradicting our assumption that x does not contain aa as subword. Analogously, if $w = bau$, then must be $u = b$ and is $x = ywwz = ybabbabz$, 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:



h. No. We already have seen that $\{ a^{2^n} \mid n \geq 0 \}$ cannot be accepted by a DFA, as an application of the pumping lemma.

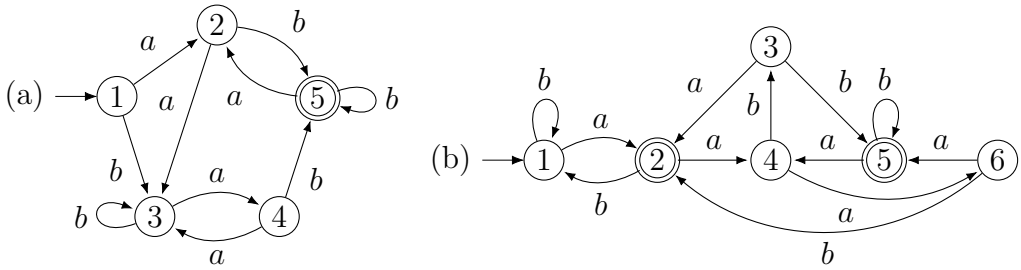
Now assume $L_h = \{ x \in \{a,b\}^* \mid n_a(x)$ is a perfect square $\}$ is accepted by a DFA. Then also $L_h \cap \{a\}^*$ is accepted by a DFA. This contradicts the above.

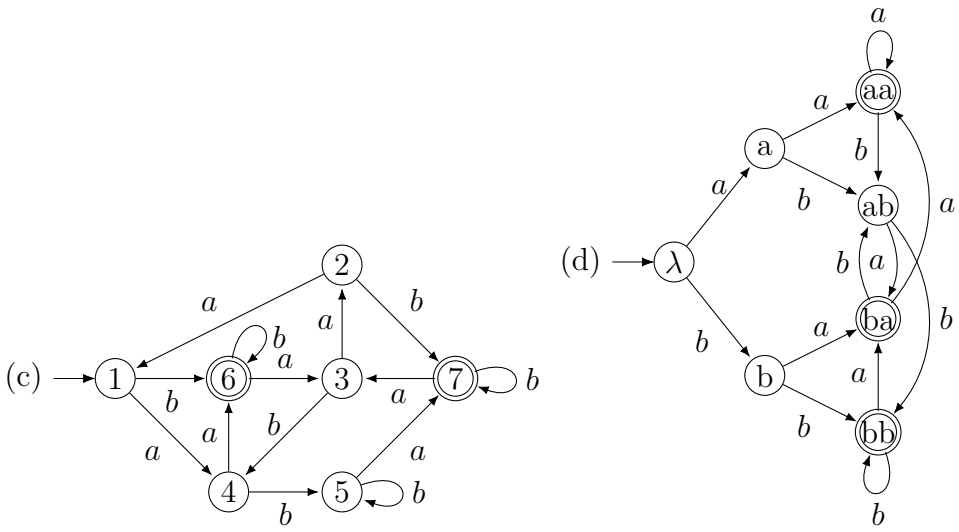
i. Yes. Keep the difference in a state. As the difference is bounded for each prefix this can be recorded in a finite number of states.

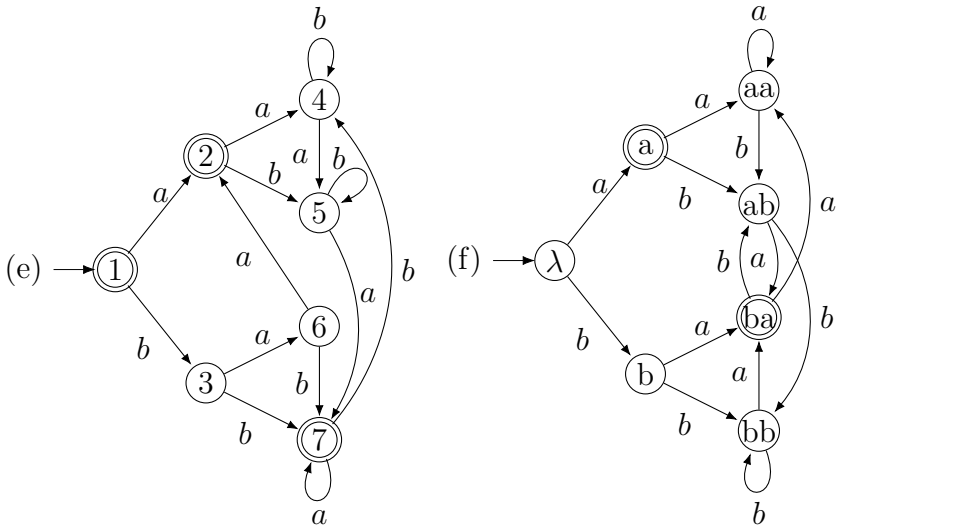
M 2.33. Let x be a string of length n in $\{a, b\}^*$, and let $L = \{x\}$. How many equivalence classes does $\equiv_L$ have? Describe them.

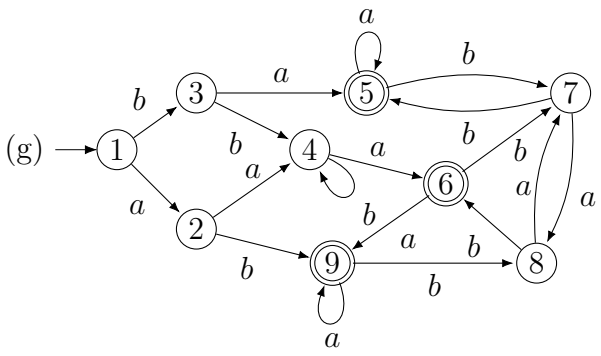
M 2.36. For a certain language $L \subseteq \{a, b\}^*$, $\equiv_L$ has exactly four equivalence classes. They are $[\lambda]$, $[a]$, $[ab]$, and $[b]$. It is also true that the three strings a, aa, and abb are all equivalent, and that the two strings b and aba are equivalent. Finally, $ab \in L$, but λ and a are not in L, and b is not even a prefix of any element of L. Draw a DFA accepting L.

M 2.55. For each of the DFAs pictured below, use the minimization algorithm to find a minimum-state DFA recognizing the same language. (It's possible that the given DFA may already be minimal.)








(g)
1
2
3
4
5
6
7
8
9
a
b

2.33 $L = \{x\}$ with x a word over $\{a, b\}$. Then $\equiv_L$ has $|x| + 2$ equivalence classes: one for each prefix of x and one containing all the words that are not a prefix of x.

For any language L, the strings that are not prefix of a string in L (if such strings exist) form a equivalence class.

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.

$\equiv_L$ 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 $\lambda, a, b \notin L$, we designate $[ab]$ as the only final state of the automaton.

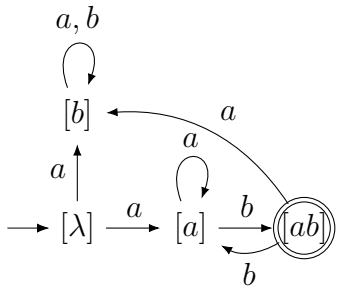
The transition function δ is defined as follows.

$\delta([\lambda], a) = [a]$ and $\delta([\lambda], b) = [b]$;

$\delta([a], b) = [ab]$ and since $a \equiv_L aa$, we set $\delta([a], a) = [aa] = [a]$;

similarly, $\delta([ab], a) = [aba] = [b]$ and $\delta([ab], b) = [abb] = [a]$.

What remains are the transitions from $[b]$. We know that b is not a prefix of any word in L. Hence (see Exercise 2.33), $[b]$ 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 $\delta([b], b) = [bb] = [b]$.



2.55 a. We construct $S = \{(p, q) \subseteq Q \times Q \mid p \not\equiv q\}$ 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.

	1	2	3	4	5
1	=				
2	2	=			
3		2	=		
4	2		2	=	
5	1	1	1	1	=

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:

	a	b
$q_0 = \{1,3\}$	q_1	q_0
$q_1 = \{2,4\}$	q_0	q_3
$q_3 = \{5\}$	q_1	q_3

Draw this DFA.

b. The given DFA is already minimal.

M 4.1. In each case below, say what language (a subset of $\{a, b\}^*$) is generated by the context-free grammar with the indicated productions.

a. $S \to aS \mid bS \mid \lambda$

b. $S \to SS \mid bS \mid a$

c. $S \to SaS \mid b$

d. $S \to SaS \mid b \mid \lambda$

e. $S \to TT \quad T \to at \mid Ta \mid b$

f. $S \to aSa \mid bSb \mid aAb \mid bAa \quad A \to aAa \mid bAb \mid a \mid b \mid \lambda$

g. $S \to aT \mid bT \mid \lambda \quad T \to aS \mid bs$

h. $S \to aT \mid bT \quad T \to aS \mid bs \mid \lambda$

4.3. In each case below, find a CFG generating the given language. The set of strings in $\{a, b\}^* \ldots$

a. $\ldots$ odd-length, with middle symbol a.

b. $\ldots$ even-length, with the two middle symbols equal.

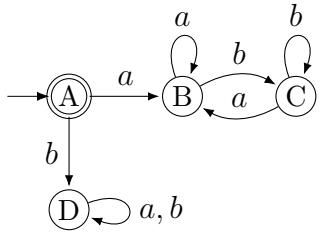
c. $\ldots$ odd-length, whose first, middle, and last symbols are all the same.

4.26. In each part, draw an NFA accepting the language generated by the CFG having the given productions.

a. $S \to aA \mid bC \quad A \to aS \mid bB \quad B \to aC \mid bA \quad C \to aB \mid bS \mid \lambda$

b. $S \to bS \mid aA \mid \lambda \quad A \to aA \mid bB \quad B \to bS \mid \lambda$

4.27. Find a right-linear grammar generating the language $L(M)$, where M is the DFA shown below.



4.29. Each of the following grammars, though not regular, generates a regular language. In each case, find a regular grammar generating the language.

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

b. $S \to AabB \quad A \to aA \mid bA \mid \lambda \quad B \to Bab \mid Bb \mid ab \mid b$

c. $S \to AAS \mid ab \mid aab \quad A \to ab \mid ba \mid \lambda$

d. $S \to AB \quad A \to aAa \mid bAb \mid a \mid b \quad B \to aB \mid bB \mid \lambda$

e. $S \to AA \mid B \quad A \to AAA \mid Ab \mid bA \mid a \quad B \to bB \mid \lambda$

4.34. Show that the CFG with productions
$S \to a \mid Sa \mid bSS \mid SSb \mid SbS$
is ambiguous.

4.36. For each part of Exercise 4.1, decide whether the grammar is ambiguous or not, and prove your answer.

4.38. In each case below, show that the grammar is ambiguous, and find an equivalent unambiguous grammar.

a. $S \to SS \mid a \mid b$

b. $S \to ABA \quad A \to aA \mid \lambda \quad B \to bB \mid \lambda$

c. $S \to aSb \mid aaSb \mid \lambda$

d. $S \to aSb \mid abS \mid \lambda$

4.48. Show that the nullable variables defined by Definition 4.7 are precisely those variables A for which $A \Rightarrow^* \lambda$. TODO

4.49. In each case below, find a context-free grammar with no λ-productions that generates the same language, except possibly for λ, as the given CFG.

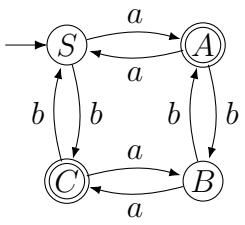
a. $S \to AB \mid \lambda \quad A \to aASb \mid a \quad B \to bS$

b. $S \to AB \mid ABC$
$A \to BA \mid BC \mid \lambda \mid a$
$B \to AC \mid CB \mid \lambda \mid b$
$C \to BC \mid AB \mid A \mid c$

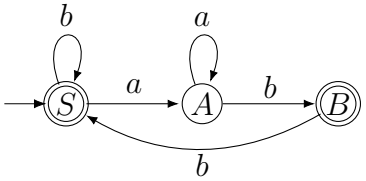
4.1 a. $L(G) = a, b^*$.
b. $L(G) = \{a, b\}^*\{a\}$.
c. $L(G) = \{ba\}^*\{b\}$.
d. $L(G) = \{x \in \{a, b\}^* \mid bb \text{ does not occur in } x\}$.
e. $L(G) = \{a\}^*\{b\}\{a\}^*\{b\}\{a\}^*$, i.e. the language of all words with exactly two occurrences of b.
f. $L(G) = \{xaybx^r, xbyax^r \mid x, y \in \{a, b\}^* \wedge y = y^r\}$, 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 $L = \{axaya, bxbyb \mid x, y \in \{a, b\}^* \wedge |x| = |y|\}$ the CFG with
$S \to aAa \mid bBb, \quad A \to aAa \mid aAb \mid bAa \mid bAb \mid a, \quad B \to aBa \mid aBb \mid bBa \mid bBb \mid b$

4.26 Describe the language generated by the given grammars.
a. $S \to aA \mid bC \mid \lambda, \quad A \to aS \mid bB, \quad B \to aC \mid bA \mid \lambda, \quad C \to aB \mid bS$
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 $L(G) = \{x \in \{a, b\}^* \mid |x| \text{ is odd }\}$.
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. $S \to bS \mid aA \mid \Lambda, \quad A \to aA \mid bB \mid b, \quad B \to bS$
This is a regular grammar. Using the construction given in the proof of Theorem 4.14, we obtain the following NFA accepting $L(G)$.



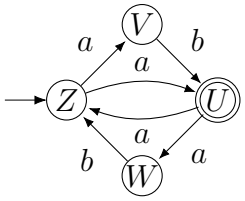
From this automaton we can read the regular expression $(b + aa^*bb)^*(\Lambda + aa^*b)$ which describes $L(G)$.

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 4.4 has the productions:
$A \to aB \mid bD \mid \lambda, \quad B \to aB \mid bC, \quad C \to aB \mid bC, \quad D \to aD \mid bD.$
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:
$A \to aB \mid \lambda, \quad B \to aB \mid bC, \quad C \to aB \mid bC.$

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 $X \to a$) generating its language.

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

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, and can be accepted by the following NFA:



A right-linear grammar for this language would be (starting symbol Z):
$Z \to aU \mid aV, \quad V \to bU, \quad U \to aZ \mid aW \mid \lambda, \quad W \to bZ$

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

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

A right-linear grammar for this language would be (with starting symbol Z):
$Z \to aZ \mid bZ \mid aU, \quad U \to bV, \quad V \to aW \mid bV \mid bX, \quad W \to bV \mid bX, \quad X \to \lambda.$

c. $S \to AAS \mid ab \mid aab, \quad A \to ab \mid ba \mid \lambda$

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 λ. 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 λ.

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

A right-linear grammar for this language would be (with starting symbol Z):
$Z \to aY \mid bX, \quad X \to aZ, \quad Y \to bZ \mid bU \mid aW, \quad W \to bU, \quad U \to \lambda.$

d. $S \to AB, \quad A \to aAa \mid bAb \mid a \mid b, \quad B \to aB \mid bB \mid \Lambda$

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 right-linear grammar for this (easy) language would be:
$Z \to aZ \mid bZ \mid aU \mid bU \quad U \to \lambda.$

e. $S \to AA \mid B, \quad A \to AAA \mid Ab \mid bA \mid a, \quad B \to bB \mid b$

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 $S \to AA$. Due to $A \to AAA$ we can generate any even (non-zero) number of A's Each A can surround itself with any number of b's before either terminating as a. Hence after $S \Rightarrow AA$ we can produce any word over $\{a, b\}$ with an even (non-zero) number of a's. Remember that we can also generate $\{b\}^+$, the (non empty) strings with zero a's. This implies that $L(G) = (\{b\}^*\{a\}\{b\}^*\{a\}\{b\}^*)^* \setminus \{\lambda\}$.

A right-linear grammar for this language would be (starting symbol Z):
$Z \to aX \mid bY, \quad X \to bX \mid aY, \quad Y \to bY \mid aX \mid \lambda.$

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 $S \Rightarrow Sa \Rightarrow SbSa \Rightarrow abSa \Rightarrow abaa$.

4.36 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.

Grammars **a.** and **h.** are both not ambiguous, as it can be proved in a similar manner as for grammar **g**.

b. The grammar 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 grammars 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 $x = \lambda$. λ 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 λ 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 λ from S, because all other productions introduce terminals. Further, λ 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.

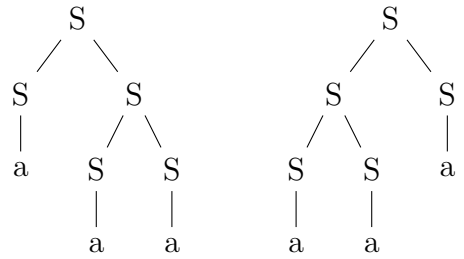
4.38 We have to show that a given grammar is ambiguous and we have to give a non-ambiguous grammar generating the same language.

a. $S \to SS \mid a \mid b$

According to this grammar the string aba has two different leftmost derivations: $S \Rightarrow SS \overset{\ell}{\Rightarrow} aS \Rightarrow aSS \Rightarrow abS \Rightarrow aba$ 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 λ, all strings over $\{a, b\}$ can be generated, that is de regular language $\{a, b\}^+$.

An equivalent regular grammar is then : $S \to aX \mid bX \quad X \to aX \mid bX \mid \lambda$.

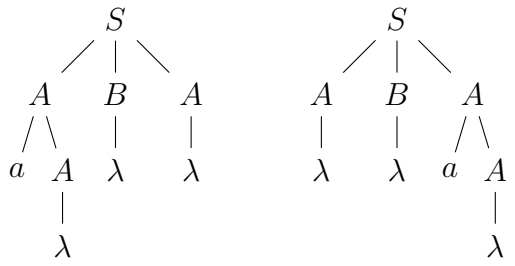
This grammar is not ambiguous because it is a regular grammar stemming from a deterministic finite automaton.

b. $S \to ABA \quad A \to aA \mid \lambda \quad B \to\mid bB \mid \lambda$

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, ie the language denoted by the regular expression $a^*b^*a^*$. An equivalent regular grammar is:

$S \to aS \mid bX \mid \lambda \quad X \to bX \mid aY \mid \lambda \quad Y \to aY \mid \lambda$.

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

d. $S \to aSb \mid aaSb \mid \lambda a$.

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 some b's where the number of a's is at least as large as the number of b 's but more than twice as large. i.e. the language $\{a^i b^j \mid j \geq i \geq 2j\}$.

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 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:

$S \to aSb \mid \lambda \mid aaAb \quad A \to aaAb \mid \lambda$.

This grammar in not ambiguous, because the only derivation of each string of the form $a^{j+k}b^j$, where $0 \leq k \leq j$, is

$S \Rightarrow^j a^j S b^j \Rightarrow a^j b^j$ if $k = 0$ and
$S \Rightarrow^{j-k} a^{j-k} S b^{j-k} \Rightarrow a^{j-k} aaAbb^{j-k} \Rightarrow^{k-1} a^{j-k+2} a^{2(k-1)} A b^{k-1} b^{j-k+1} \Rightarrow a^{j+k} b^j$ if $k \geq 1$.

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 righthand-side λ 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 $A \to B_1 B_2 \dots B_n$ 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 $A \Rightarrow B_1 B_2 \dots B_n \Rightarrow^* B_2 \dots B_n \Rightarrow^* B_n \Rightarrow^* \lambda$ 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. Next assume (induction hypothesis) that whenever $B \Rightarrow^k \lambda$ for some $k \leq m$, then B is nullable. Then consider the case $A \Rightarrow^{m+1} \lambda$. 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 $A \Rightarrow B_1 \dots B_n \Rightarrow^m \lambda$. Consequently, for each $1 \leq i \leq n$, we have $B_i \Rightarrow^{k_i} \lambda$ 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 λ-productions that generates the same language (except for λ) as the given CFG. We apply Algorithm 6.1.

a. CFG G is given as $S \to AB \mid \lambda, \quad A \to aASb \mid a, \quad B \to bS$.

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

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 λ-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 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
$S \to AB \mid ABC, \quad A \to BA \mid BC \mid \lambda \mid a,$
$B \to AC \mid CB \mid \lambda \mid b, \quad C \to BC \mid AB \mid A \mid c.$
The nullable variables are obtained as $N_3 = N_2 = \{S, A, B, C\}$ from
$N_0 = \{A, B\}$, $N_1 = N_0 \cup \{C\}$, $N_2 = N_1 \cup \{S\}$.
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 Λ 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 $L(G) - L(G') = \{\lambda\}$.

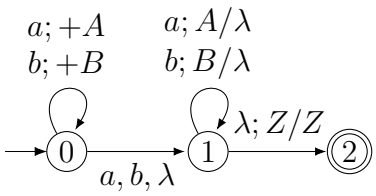
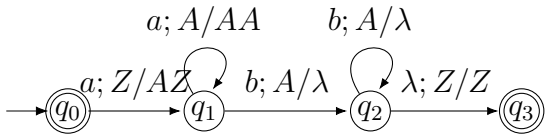


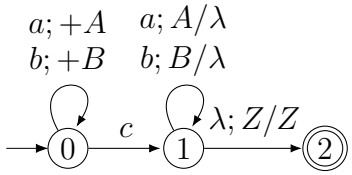
Figure 1: Automaton for Pal (5.4) (5.5)

5.1.

a. For the PDA with language AnBn below, trace the sequence of moves made for the input strings *ab*, *aab*, and *abb*. Initial stack symbol Z.



b. For the PDA with language SimplePal below, trace the sequence of moves made for the input strings *bacab* and *baca*. Initial stack symbol Z.



5.3. For an input string $x \in \{a, b\}^*$ with $|x| = n$, how many possible complete sequences of moves (i.e., start in the initial configuration (q_{in}, x, Z_{in}) and terminate in a configuration from which no move is possible) can the PDA for Pal in Figure 1 make? It is helpful to remember that once the PDA reaches the state 1, there is no choice of moves.

5.4. Consider again the PDA for Pal, given in Figure 1. For each of the following languages over $\{a, b\}$, modify it to obtain a PDA accepting the language.

a. The language of even-length palindromes.

b. The language of odd-length palindromes.

5.6. In both cases below, a transition table is given for a PDA with initial state q_0 and accepting state q_2. Describe in each case the language that is accepted.

a. 1. $(q_0, a, Z) \mapsto (q_1, aZ)$
   2. $(q_0, b, Z) \mapsto (q_1, BZ)$
   3. $(q_1, a, A) \mapsto (q_1, A)$, $(q_1, a, A) \mapsto (q_2, A)$
   4. $(q_1, b, A) \mapsto (q_1, A)$
   5. $(q_1, a, B) \mapsto (q_1, B)$
   6. $(q_1, b, B) \mapsto (q_1, B)$, $(q_1, b, B) \mapsto (q_2, B)$

b. 1. $(q_0, \sigma, Z) \mapsto (q_0, XZ)$, $\sigma \in \{a, b\}$
   3. $(q_0, \sigma, X) \mapsto (q_1, XX)$, $\sigma \in \{a, b\}$

5. $(q_0, c, A) \mapsto (q_1, X)$, $A \in \{X, Z\}$
7. $(q_1, \sigma, X) \mapsto (q_1, \lambda)$, $\sigma \in \{a, b\}$
9. $(q_1, \lambda, Z) \mapsto (q_2, Z)$

5.10. Show that every regular language can be accepted by a deterministic PDA M with only two states in which there are no λ-transitions and no symbols are ever removed from the stack.

5.13. Suppose $L \subseteq \Sigma^*$ is accepted by a PDA M, and for some fixed k and every $x \in L$, no sequence of moves made by M on input x causes the stack to have more than k elements. Show that L is regular.

5.15. Suppose $L \subseteq \Sigma^*$ is accepted by a PDA M, and for some fixed k, and every $x \in L$, at least one choice of moves allows M to accept x in such a way that the stack never contains more than k elements. Does it follow that L is regular? Prove your answer.

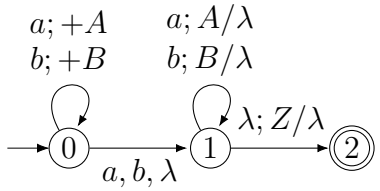
5.28. In each case below, you are given a CFG G and a string x that it generates. For the top-down PDA $NT(G)$, trace a sequence of moves by which x is accepted, showing at each step the state, the unread input, and the stack contents. Show at the same time the corresponding leftmost derivation of x in the grammar.

a. The grammar has productions $S \to S + T \mid T \quad T \to T * F \mid F \quad F \to [\, S\,] \mid a$ and $x = [\, a + a * a\,] * a$.

b. The grammar has productions $S \to S + S \mid S * S \mid [\, S\,] \mid a$, and $x = [\, a * a + a\,]$.

c. The grammar has productions $S \to [\, S\,]\, S \mid \lambda$, and $x = [\,]\,[\,[\,]\,[\,]\,]$.

5.32. Let M be the PDA given for Pal, except that the final move is changed to $(2, \lambda)$, so that M does in fact accept by empty stack. See below.
Let $x = ababa$. Find a sequence of moves of M by which x is accepted, and give the corresponding leftmost derivation in the CFG obtained from M as in Theorem 5.29.



5.34. In each case below, you are given a CFG G and a string x that it generates. For the nondeterministic bottom-up PDA NB(G), trace a sequence of moves by which x is accepted, showing at each step the stack contents and the unread input. Show at the same time the corresponding rightmost derivation of x (in reverse order) in the grammar.

a. The grammar has productions $S \to S\,[\, S\,] \mid \lambda$, and $x = [\,]\,[\,[\,]\,]$.

b. The grammar has productions $S \to [\, S\,]\, S \mid \lambda$, and $x = [\,]\,[\,[\,]\,]$.

5.1.a $(q_0, ab, Z_0) \vdash (q_1, b, aZ_0) \vdash (q_2, \lambda, Z_0) \vdash$, $(q_3, \lambda, Z_0) \vdash$, acceptance.
$(q_0, aab, Z_0) \vdash (q_1, ab, aZ_0) \vdash (q_1, b, aaZ_0) \vdash (q_2, \lambda, aZ_0) \vdash$, crash.
$(q_0, abb, Z_0) \vdash (q_1, bb, aZ_0) \vdash (q_2, b, Z_0) \vdash$, crash.
5.1.b $(q_0, bbcbb, Z_0) \vdash (q_0, bcbb, bZ_0) \vdash (q_0, cbb, bbZ_0) \vdash (q_1, bb, bbZ_0) \vdash (q_1, b, bZ_0) \vdash (q_1, \lambda, Z_0)$ $\vdash (q_2, \lambda, Z_0)$, acceptance.
$(q_0, baca, Z_0) \vdash (q_0, aca, bZ_0) \vdash (q_0, ca, abZ_0) \vdash (q_1, a, abZ_0) \vdash (q_1, \lambda, bZ_0) \nvdash$, 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 λ-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 $2n + 1$ 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 λ-move have been removed.

5.6 a. This PDA accepts $\{axa, bxb \mid x \in \{a, b\}^*\}$.
b. The PDA accepts $\{xcy \mid x, y \in \{a, b\}^*$ and $|x| = |y|\}$.

5.10 Let $M_0 = (Q, \Sigma, q_0, A, \delta)$ be an DFA. 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 $r \notin A$.
Note that M never removes a symbol from the stack, has no λ-transitions, and is deterministic (because M_0 is deterministic).

5.28 b. Given is the CFG with productions $S \to S + S \mid S * S \mid (S) \mid a$. This grammar generates the string $x = (a * a + a)$. 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.

state	string	stack		step
q_0	$(a*a+a)$	Z_0		
q_1	$(a*a+a)$	SZ_0		S
q_1	$(a*a+a)$	$(S)Z_0$		$\Rightarrow (S)$
q_1	$a*a+a)$	$S)Z_0$		
q_1	$a*a+a)$	$S+S)Z_0$		$\Rightarrow (S+S)$
q_1	$a*a+a)$	$S*S+S)Z_0$		$\Rightarrow (S*S+S)$
q_1	$a*a+a)$	$a*S+S)Z_0$		$\Rightarrow (a*S+S)$
q_1	$*a+a)$	$*S+S)Z_0$		
q_1	$a+a)$	$S+S)Z_0$		
q_1	$a+a)$	$a+S)Z_0$		$\Rightarrow (a*a+S)$
q_1	$+a)$	$,+S)Z_0$		
q_1	$a)$	$S)Z_0$		
q_1	$a)$	$a)Z_0$		$\Rightarrow (a*a+a)$
q_1	$)$	$)Z_0$		
q_1	λ	Z_0		
q_2	λ	Z_0		

5.32 Consider the PDA M from Example 5.7 with move 12 $\delta(q_1, \lambda, Z_0) = \{(q_2, Z_0)\}$ changed into $\delta(q_1, \lambda, Z_0) = \{(q_2, \lambda)\}$.

state	string	stack		step
q_0	$ababa$	Z_0		$S \Rightarrow [q_0, Z_0, q_2]$
q_0	$baba$	aZ_0		$\Rightarrow a[q_0, a, q_1][q_1, Z_0, q_2]$
q_0	aba	baZ_0		$\Rightarrow ab[q_0, b, q_1][q_1, a, q_1][q_1, Z_0, q_2]$
q_1	ba	baZ_0		$\Rightarrow aba[q_1, b, q_1][q_1, a, q_1][q_1, Z_0, q_2]$
q_1	a	aZ_0		$\Rightarrow abab[q_1, a, q_1][q_1, Z_0, q_2]$
q_1	λ	Z_0		$\Rightarrow ababa[q_1, Z_0, q_2]$
q_2	λ	λ		$\Rightarrow ababa$

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.

5.34 a. Given is the CFG with productions $S \to [S]S \mid \lambda$. This grammar generates the string $x = [][[]]$. We consider the bottom-up PDA constructed from the grammar as in Example 5.24 and trace a sequence of steps by which x is accepted together with the corresponding rightmost derivation of x in the grammar in reverse order.

move	state	string	stack		step
	q	$[][]$	Z_0		
reduce	q	$[][]$	SZ_0		$\Rightarrow [][]$
shift	q	$][]$	$[SZ_0$		
reduce	q	$][]$	$S[SZ_0$		$\Rightarrow S[][]$
shift	q	$[]$	$]S[SZ_0$		
reduce	$q_{1,1}$	$[]$	$S[SZ_0$		$\Rightarrow S[S][]$
	$q_{1,2}$	$[]$	$[SZ_0$		
	$q_{1,3}$	$[]$	SZ_0		
	q	$[]$	SZ_0		
shift	q	$]$	$[SZ_0$		
reduce	q	$]$	$S[SZ_0$		$\Rightarrow S[]$
shift	q	$]]$	$[S[SZ_0$		
reduce	q	$]]$	$S[S[SZ_0$		$\Rightarrow S[S[]]$
shift	q	$]$	$]S[S[SZ_0$		
reduce	$q_{1,1}$	$]$	$S[S[SZ_0$		$\Rightarrow S[S[S]]$
	$q_{1,2}$	$]$	$[S[SZ_0$		
	$q_{1,3}$	$]$	$S[SZ_0$		
	q	$]$	$S[SZ_0$		
shift	q	λ	$]S[SZ_0$		
reduce	$q_{1,1}$	λ	$S[SZ_0$		$S \Rightarrow S[S]$
	$q_{1,2}$	λ	$[SZ_0$		
	$q_{1,3}$	λ	SZ_0		
	q	λ	SZ_0		
(pop S)	q_1	λ	Z_0		
(accept)	q_2	λ	Z_0		

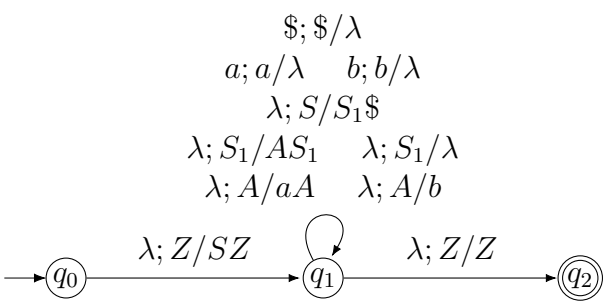
5.36 Let M be a PDA (accepting with empty stack) and consider the CFG G obtained from M as in the proof of Theorem 5.29. Since every accepting computation of M determines a unique leftmost derivation of a word in $L(G)$, it follows that G is unambiguous if M is deterministic.

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:

$S \to S_1\$, \quad S_1 \to AS_1 \mid \lambda, \quad A \to aA \mid b.$

The top-down PDA associated with G is given below through its transition diagram.



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

6.2. In each case below, show using the pumping lemma that the given language is not a CFL.

a. $L = \{\ a^i b^j c^k \mid i < j < k\ \}$

b. $L = \{\ a^{2^n} \mid n \geq 0\ \}$

c. $L = \{\ x \in \{a,b\}^* \mid n_b(x) = n_a(x)^2\ \}$

d. $L = \{\ a^n b^{2n} a^n \mid n \geq 0\ \}$

e. $L = \{\ x \in \{a,b,c\}^* \mid n_a(x) = \max\{n_b(x), n_c(x)\}\ \}$

f. $L = \{\ x \in \{a,b,c\}^* \mid n_a(x) = \min\{n_b(x), n_c(x)\}\ \}$

g. $\{\ a^n b^m a^n b^{n+m} \mid m,n \geq 0\ \}$

6.5. For each case below, decide whether the given language is a CFL, and prove your answer.

a. $L = \{\ a^n b^m a^m b^n \mid m,n \geq 0\ \}$

b. $L = \{xayb \mid x,y \in \{a,b\}^*$ and $|x| = |y|\ \}$

c. $L = \{xcx \mid x \in \{a,b\}^*\ \}$

d. $L = \{xyx \mid x,y \in \{a,b\}^*$ and $|x| \geq 1\ \}$

e. $L = \{x \in \{a,b\}^* \mid n_a(x) < n_b(x) < 2n_(x)\ \}$

f. $L = \{x \in \{a,b\}^* \mid n_a(x) = 10\, n_b(x)\ \}$

g. L is the set of non-balanced strings of parentheses

6.12.

a. Show that if L is a CFL and F is finite, $L - F$ is a CFL.

b. Show that if L is not a CFL and F is finite, then $L - F$ is not a CFL.

c. Show that if L is not a CFL and F is finite, then $L \cup F$ is not a CFL.

6.13. For each part of Exercise 6.12, say whether the statement is true if "finite" is replaced by "regular", and give reasons.

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 $vw^i xy^i z \in L$ 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^2xy^2z contains at least $n+1$ a's and exactly $n+2$ c's, which implies that vw^2xy^2z 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^0xy^0z contains n a's and either at most n b's or at most $n+1$ c's. Thus $vw^0xy^0z \notin L$, 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.

c. $L = \{x \in \{a, b\}^* \mid n_b(x) = n_a(x)^2\}$.

Examples of words in L are: *aabbbb*, *babbba*, *abbbabbbabbb*.

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

Let $n_a(wy) = p$ and $n_b(wy) = q$. Then for each $i \geq 0$ we have $n_a(vw^i xy^i z) = n_a(u) + (i-1)p = n + (i-1)p$ and $n_b(vw^i xy^i z) = n_b(u) + (i-1)q = n^2 + (i-1)q$. Since, by our assumption $vw^i xy^i z \in L$ for every $i \geq 0$, it must be the case that $(n + (i-1)p)^2 = n^2 + (i-1)q$ for every $i \geq 0$. This however is impossible as can be seen as follows. Since $|wy| > 0$, at least one of p and q is not 0.

If $p = 0$ and $q \neq 0$, then $n^2 = n^2 + (i-1)q$ for every $i \geq 0$, which clearly is not true if $i \geq 2$.

If $p \neq 0$ and $q = 0$, then $(n + (i-1)p)^2 = n^2$ for every $i \geq 0$, which clearly is not true if $i \geq 2$.

If $p \neq 0$ and $q \neq 0$, then we have (for $i = 2$) that $(n+p)^2 = n^2 + q$ which implies that $q = 2np + p^2$, and (for $i = 3$) that $(n+2p)^2 = n^2 + 2q$ which implies that $q = 2np + 2p^2$. Hence $p = 0$ should hold, a contradiction.

We conclude that the pumping lemma is not satisfied and that, consequently, L is not context-free.

6.5 Is the given language context-free? Prove your answer.

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

$$S \to Yb \quad Y \to aSX \mid bSX \mid a \quad X \to a|b\,.$$

c. $L = \{xcx \mid x \in \{a, b\}^*\}$ is not a CFL; proof similar as in Example 6.2.

d. $L = \{xyx \mid x, y \in \{a, b\}^*$ 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 $pq^i rs^i t \in L$ 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 pq^2rs^2t is either $a^{n+j}b^na^nb^n$ or $a^nb^na^nb^{n+j}$ 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 pq^0rs^0t is either $a^kb^la^mb^n$ or $a^nb^ka^lb^m$ 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 CFLand F a regular language. Then $L - F = L \cap \bar{F}$ is a CFL, because the complement $\bar{F}$ 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 Σ is an alphabet such that $L \subseteq \Sigma^*$, then $L - F = \emptyset$ which is a CFL.