Automata Theory

Answer to selected exercises 2

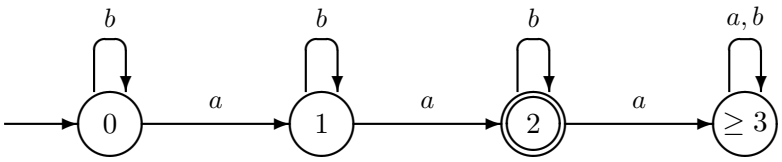
John C Martin: Introduction to Languages and the Theory of Computation
Fourth edition

Rudy van Vliet (and Marcello Bonsangue and Jetty Kleijn)

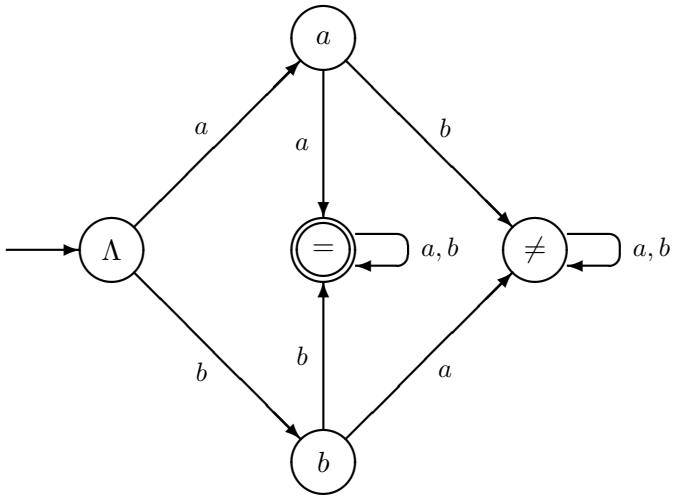
Fall 2024

2.1

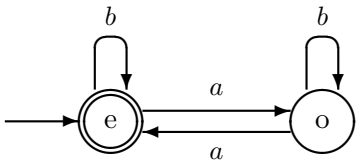
a. States count the number of a's read so far.



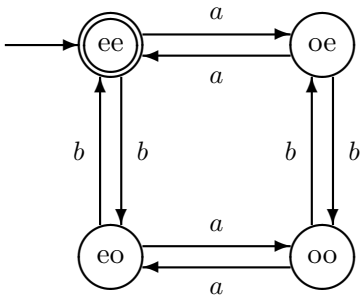
d. States keep track of the first two characters read.



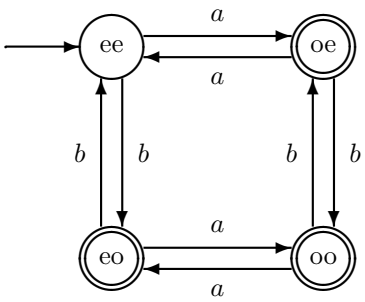
f. States represent whether the number of a's read so far is even or odd



g. States represent whether the number of a's read so far is even or odd and whether the number of b's read so far is even or odd.



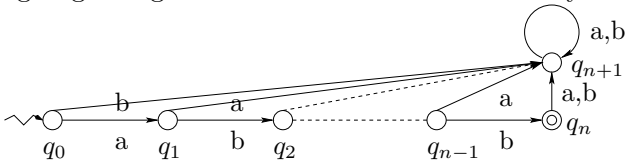
g2. When you consider the language of strings over $\{a, b\}$ in which either the number of a's or the number of b's is odd (or both), you can take almost the same finite automaton as in part **g**. Only the accepting states and the nonaccepting states must be reversed, because the language is the complement of the language from part **g**.



2.2

(a) All strings containing a substring $aaba$.
(b) All strings ending with $aaba$.
(c) All strings starting with $aaba$.
(d) The empty string, and all strings starting with a and ending with b.
(e) All strings resulting when you concatenate zero or more strings from $\{ab, ba\}$ (the same string may be chosen multiple times).

2.3 b Let $x \in \{a, b\}^*$ be such that $|x| = n$ for some $n \geq 0$. Thus $x = a_1 a_2 \ldots a_n$ with each $a_i \in \{a, b\}$. Then there is a finite automaton accepting $\{x\}$ with $n + 2$ states, namely initial state q_0, intermediate states q_1, ..., q_{n-1}, accepting state q_n, and sink state q_{n+1}. Observe that for $x = \Lambda$ we have $q_0 = q_n$ as both the initial and the accepting state. Each non-sink state q_i represents the prefix of length i of x and has a transition to the state representing the next longer prefix and — for the "wrong" input symbol a transition to the sink state; from the state q_n representing x and from the sink itself there are only transitions to the sink. Below is an example for a string beginning with ab and with b as its last symbol.



Observe that $n + 2$ states are necessary because different prefixes need different states: if $x = yz$ and $x = uv$ with y and u two different prefixes of x, then $uz \neq yz = x$ and hence uz should not be accepted, which implies that u should be distinguished from y. That is, y and u should lead to different states. Since x has $n+1$ different prefixes this implies that we need at least $n + 1$ states in any finite automaton accepting $\{x\}$. Moreover, all prefixes have to be distinguished from all non-prefixes of x since no string starting with a non-prefix of x will ever lead to acceptance.

2.5 Let $M = (Q, \Sigma, q_0, A, \delta)$ be an FA. We prove by induction on $|y|$ that for all $x, y \in \Sigma^*$ and for all $q \in Q$,
$\delta^*(q, xy) = \delta^*(\delta^*(q, x), y)$.
Let $x \in \Sigma^*$ be an arbitrary string.
basis: $|y| = 0$, that is, $y = \Lambda$. Since, by definition $\delta^*(p, \Lambda) = p$ for all states p, it follows that $\delta^*(q, xy) = \delta^*(q, x) = \delta^*(\delta^*(q, x), \Lambda) = \delta^*(\delta^*(q, x), y)$ as required.
induction hypothesis: there is a $k \geq 0$ such that for all $z \in \Sigma^*$ with $|z| \leq k$, and for all $q \in Q$, $\delta^*(q, xz) = \delta^*(\delta^*(q, x), z)$.
induction step: let $|y| = k + 1$, that is, $y = za$ for some $a \in \Sigma$ and $z \in \Sigma^*$ such that $|z| = k$.
We now have $\delta^*(q, xy) = \delta^*(q, xza) = \delta(\delta^*(q, xz), a)$, by the definition of δ^*. With the induction hypothesis we obtain $\delta(\delta^*(q, xz), a) = \delta(\delta^*(\delta^*(q, x), z), a)$ which — again by the definition of δ^* — equals $\delta^*(\delta^*(q, x), za) = \delta^*(\delta^*(q, x), y)$.
Our claim now follows for the chosen x, but since that was an arbitrary string, the statement has been proved for all x.

Using induction on $|x|$ rather than on $|y|$ would have been awkward (or even impossible) given the structure of the (inductive) definition of δ^*.

2.6 Let $M = (Q, \Sigma, q_0, A, \delta)$ be an FA. Let $q \in Q$ be such that $\delta(q, a) = q$ for all $a \in \Sigma$. This means that whatever the input symbol, if M is in state q it will remain there. We use induction to prove that once in q, M will never leave that state anymore whatever the input string it is being fed.
By definition, we have that $\delta((q, \Lambda) = q$.
Now assume that there is an integer $k \geq 0$ such that $\delta^*(q, z) = q$, for all $z \in \Sigma^*$ with $|z| \leq k$. Let $x \in \Sigma^*$ be of length $k + 1$. So, $x = za$ for some $a \in \Sigma$ and $z \in \Sigma^*$ with $|z| = k$. Consequently, we can use the induction hypothesis and deduce that $\delta^*(q, x) = \delta^*(q, za) = \delta(\delta^*(q, z), a) = \delta(q, a) = q$ by the property given for q.

• *Show that if for some state q and some string x, $\delta^*(q, x) = q$, then for every $n \geq 0$, $\delta^*(q, x^n) = q$.*
Let $q \in Q$ and $x \in \Sigma^*$ be such that $\delta(q, x) = q$. Thus, if M is in state q it will always return to q if it reads x. We use induction on n to prove that M will always return to q whatever the number n of times it has read x starting in q: $\delta^*(q, x^n) = q$ for all $n \geq 0$.
Clearly, $\delta^*(q, x^0) = \delta^*(q, \Lambda) = q$.
Assume we have proved the statement for some integer $k \geq 0$.
Now consider $n = k + 1$. Then $\delta^*(q, x^n) = \delta^*(q, x^{k+1}) = \delta^*(\delta^*(q, x^k), x)$ by Exercise 2.5. Using the induction hypothesis we obtain $\delta^*(\delta^*(q, x^k), x) = \delta^*(q, x) = q$ by the assumed property of q.

• *Let $M = (Q, \Sigma, q_0, A, \delta)$ be an FA and let $M_1 = (R, \Sigma, q_0, A \cap R, \delta_1)$ be the FA obtained from M by deleting those states $q \in Q$ that are not reachable from q_0 together with all transitions leading to and from them (i.e there is no string $x \in \Sigma^*$ such that $\delta^*(q_0, x) = q$) Show that $L(M_1) = L(M)$.*
Take $R = \{p \in Q \mid \exists x \in \Sigma^* \text{ such that } \delta^*(q_0, x) = p\}$. It consists of all states in Q reachable from q_0. Let δ_1 is the restriction of δ to R, that is, $\delta_1(p, a) = \delta(p, a)$ for all $p \in R$ and all $a \in \Sigma$.
Since M_1 is a "sub-automaton" of M it follows immediately that $L(M_1) \subseteq L(M)$. To prove the converse inclusion we consider an arbitrary string $w \in L(M)$. Let $p \in A$ be the accepting state such that $\delta^*(q_0, w) = p$. Hence p is reachable: $p \in A \cap R$. Moreover, every state $s \in Q$ such that $\delta*(q_0, v) = s$ for some prefix of w, is reachable from q_0 and hence in R and all its transitions are in δ_1. Consequently, $\delta^*(q_0, w) = \delta_1^*(q_0, w) = p \in A \cap R$ which implies that $w \in L(M_1)$.

2.10 a., b. and c. The desired automata each have (A, X) as initial state. Next we apply the product construction to M_1 and M_2:

	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)

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.
For $L_1 \cup L_2$, we have as accepting states $\{(C, Y), (C, Z), (A, Z), (B, Z)\}$, that is any pair of original states in which at least one is accepting.
For $L_1 \cap L_2$, we have as accepting states $\{(C, Z)\}$, that is any pair of original states in which both are accepting.
For $L_1 - L_2$, we have as accepting states $\{(C, Y)\}$, that is any pair of original states in which the first is accepting and the second not.
Drawing the automata is now easy.

2.11 Let $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$ be an FA and let $M_2 = (Q_2, \Sigma, q_2, A_2, \delta_2)$. Define for all $p \in Q_1$, $q \in Q_2$, and $a \in \Sigma$: $\delta((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$.
We have to prove that $\delta^*((p, q), x) = (\delta_1^*(p, x), \delta_2^*(q, x))$ for all $p \in Q_1$, $q \in Q_2$, and $x \in \Sigma^*$ (see the proof of Theorem 2.15).
We use induction on $|x|$, the length of x.
If $|x| = 0$, then $x = \Lambda$ and we have $\delta^*((p, q), \Lambda) = (p, q) = (\delta_1^*(p, \Lambda), \delta_2^*(q, \Lambda))$ by the inductive definitions of δ^*, δ_1^*, and δ_2^*.
Let $|x| = n+1$. Then $x = ya$ for some $y \in \Sigma^*$ and $a \in \Sigma$. Hence $|y| = n$ and according to the induction hypothesis: $\delta^*((p, q), y) = (\delta_1^*(p, y), \delta_2^*(q, y))$.
So, $\delta^*((p, q), x) = \delta^*((p, q), ya) = \delta(\delta^*((p, q), y)a)$ by the inductive definition of δ^*. By the induction hypothesis $\delta(\delta^*((p, q), y), a) = \delta((\delta_1^*(p, y), \delta_2^*(q, y)), a)$ and $\delta((\delta_1^*(p, y), \delta_2^*(q, y)), a) = (\delta_1((\delta_1^*(p, y), a), \delta_2((\delta_2^*(q, y), a))$ by the definition of δ. Finally, using the inductive definition of δ_1^* and δ_2^*, we derive $(\delta_1((\delta_1^*(p, y), a), \delta_2((\delta_2^*(q, y), a)) = (\delta_1^*(p, ya), \delta_2^*(q, ya)) = (\delta_1^*(p, x), \delta_2^*(q, x))$ as desired.

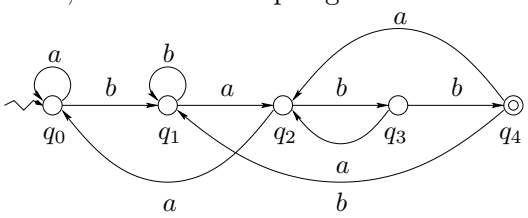
• *Show by an example that for some language L, any FA recognizing L must have more than one accepting state. Characterize those languages for which this is true.*

Let $L = \{\Lambda, 0\} \subseteq \{0\}^*$. Observe that Λ and 0 are distinguishable with respect to L, since they are distinguished by, e.g., the string $z = 0$: $\Lambda 0 \in L$ but 00 is not a string from L. Thus any FA accepting L has two distinct states (see Lemma 3.1.) for Λ and 0. Since both strings are in L, these states are both accepting.
In general, for every language which contains at least two distinguishable strings, any FA recognizing that language has at least two accepting states. This can — similar to the above — be seen as follows: Let M be an arbitrary FA accepting L and let $u, v \in L$ be distinguishable with respect to L. Then by Lemma 3.1, $\delta^*(q_0, u) \neq \delta^*(q_0, v)$ where q_0 is the initial state of M. Consequently, M has at least two accepting states.
Also the converse holds: for every language L that can be accepted by an FA, it is the case that if no FA accepting L has less than two accepting states, then L contains two strings which are distinguishable with respect to L. This can be proved once it has been shown that for every language that can be accepted by an FA, there exists a minimal FA: two strings are indistinguishable w.r.t. the language if and only if they lead to the same state in that FA.

2.13 Consider the FA from Figure 2.17d. Note that the picture in the book is wrong (it is not deterministic!) and should be corrected by removing the b-transition from the initial state to the unique accepting state. In this case, the deterministic 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 any two of these strings are distinguishable, then it follows from Theorem 2.21 that any FA recognizing L has at least 4 states.
for Λ and a and for ab and aa, choose $z = a$:
$\Lambda a \in L$, $aba \in L$, but $aa \notin L$ and $aaa \notin L$;
for Λ and ab, for Λ and aa, for a and ab, and for a and aa, choose $z = \Lambda$:
$\Lambda\Lambda \in L$, $a\Lambda \in L$, but $ab\Lambda \notin L$ and $aa\Lambda \notin L$.
Hence, there is no FA accepting L with fewer states than the given FA.

2.14 Let z be a string over the alphabet $\{a, b\}$. Any FA which accepts $L = \{a, b\}^*\{z\}$ 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. If $z = xy$ and $z = uv$ with $|x| > |u|$, then $xy \in L$, but $uy \notin L$. Hence x and u are distinguishable with respect to L.
No more states are needed, since there is no need to distinguish between a string and the longest prefix of z which is a suffix of this string:

Consider a string x and let v be its longest suffix such that $x = uv$ where v is such that $z = vw$ for some w. (Note that *every* string x has such a suffix!) Then for all strings $y \in \{0,1\}^*$ we have
case $|y| \geq |z|$: $xy \in L$ if and only if $vy \in L$ because it is only relevant whether y ends with z or not;
or case $|y| < |z|$: then $xy \in L$ if and only if $uvy \in L$ if and only if vy ends with z if and only if $vy \in L$.
Consequently, x and v are indistinguishable and we know how to define an FA accepting L.
As an example we give an FA accepting $\{a,b\}^*\{babb\}$ 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 accepting state since its last symbols form $z = babb$.



2.15 Let $L = \{aa\}^*$, the language of all even length a-strings. Then we have infinitely many pairs of distinguishable strings: for all $i \geq 0$, the string a^{2i} and the string a^{2i+1} are distinguishable with respect to L.
Note that the language L is accepted by a FA (as is easy to see). In fact, there are only two types of indistinguishable strings with respect to L (the even length strings versus the odd length strings).

2.16 Let $n \geq 0$. Let K be an arbitrary non-empty language consisting of strings of length n only. Then any FA that accepts K has at least $n + 2$ states which can be seen as follows:
First, we observe that if $n = 0$, then it must be the case that $K = \Lambda$ and any FA accepting K has 2 states (the initial state which is also the only accepting state, and a sink state).
Next, we assume that $n \geq 1$ and we consider a (fixed) string $x \in K$. Since $K \neq \emptyset$, such x exists. This x has (at least) two different prefixes $x_1 \neq x_2$, thus: $x = x_1y_1 = x_2y_2$ for some y_1, y_2. Obviously, $x_2y_1 \notin K$ and so x_1 and x_2 can be distinguished w.r.t. K. Moreover (the infinitely many) strings which are not a prefix of a string in K can be distinguished from the $n + 1$ prefixes of x. Consequently, by Theorem 2.21, any FA accepting K has at least $n + 2$ states.
We conclude that for any infinite language $L \subseteq \{a,b\}^*$ any FA which accepts

its subset $L_n = \{x \in L \mid |x| = n\}$ has at least $n+2$ states ($s(n) = 2$). The language $L = \{a,b\}^*$ has the property that each of its L_n's can be recognized by an FA with exactly $s(n) = n+2$ states. Any language that does not contain two strings of the same length also has this property.

2.17 Let $L = \{a^n b^n \mid n \geq 0\}$.
a. Give strings $x, y \in \{a,b\}^*$ such that $x \neq y$ and $xI_L y$ (that is x and y are indistinguishable w.r.t. L).
$x = a$ and $y = ba$: both are strings not in L and both are not prefixes of any string in L. This means that for all $z \in \{a,b\}^*$ we have $xz \notin L$ and $yz \notin L$. In general: every pair of strings which both are not a prefix of a string in L are indistinguishable.
$x = ab$ and $y = aabb$: both are in L and any non-empty string concatenated to them yields a string not in L. This means that $x\Lambda \in L$ and $y\Lambda \in L$ and $xz \notin L$ and $yz \notin L$ for all non-empty strings z. Hence, for all $z \in \{a,b\}^*$ we have $xz \in L$ if and only if $yz \in L$ and so x and y are indistinguishable with respect to L. In general: every pair of non-empty strings of L are indistinguishable with respect to L. Note that $\Lambda \in L$ can be distinguished from $x = ab$, because for example $\Lambda a^3 b^3 \in L$ but $aba^3 b^3 \notin L$.
$x = aab$ and $y = a^3 b^2$: both are not in L, adding a single b gives for each a string in L, while all other strings will lead to a string not in L. Hence we have $x\Lambda \notin L$ and $y\Lambda \notin L$, $xb \in L$ and $yb \in L$, and $xz \notin L$ and $yz \notin L$ for all $z \in \{a,b\}^*$ with $z \neq \Lambda$ and $z \neq b$. Consequently, for all $z \in \{a,b\}^*$ we have $xz \in L$ if and only if $yz \in L$. In general: every pair $x = a^m b^{m-k}$ and $y = a^n b^{n-k}$ with $n \neq m$ and $n > k$ or $m > k$ are indistinguishable with respect to L.
b. Consider a^m and a^n with $m \neq n$. These strings can be distinguished w.r.t. L (that is $a^m I_L a^n$ does not hold), because $a^m b^m \in L$, but $a^n b^m \notin L$. From Theorem 2.26 we conclude that any finite automaton recognizing L needs a separate state $\delta^*(q_0, a^k)$ for all $k \geq 0$, a contradiction with an FA having only finitely many states. Thus we conclude that L cannot be recognized by a FA.

2.21 In each case below, a language $L \subseteq \{a,b\}^*$ is given and we have to prove that the elements of the infinite set $\{a^n \mid n \geq 0\}$ are pairwise L-distinguishable. (which implies that L is not regular according to Theorem 2.26).
a. $L = \{a^n b a^{2n} \mid n \geq 0\}$; the strings a^i and a^j with $0 \leq i < j$ are distinguished by ba^{2i}, because $a^i b a^{2i} \in L$ and $a^j b a^{2i} \notin L$.
b. $L = \{a^k b^l a^m \mid m > k + l \geq 0\}$; the strings a^i and a^j with $0 \leq i < j$ are

distinguished by ba^{i+2}, because $a^iba^{i+2} \in L$ and $a^jba^{i+2} \notin L$.
c. $L = \{a^kb^l \mid l = k \vee l = 2k \geq 0\}$; the strings a^i and a^j with $0 \leq i < j$ are distinguished by b^{2j}, because $a^ib^{2j} \notin L$ and $a^jb^{2j} \in L$.
d. $L = \{a^kb^l \mid l \text{ is a multiple of } k\}$; the strings a^i and a^j with $0 \leq i < j$ are distinguished by ab^{i+1}, because $a^iab^{i+1} \in L$ and $a^jab^{i+1} \notin L$ (why can't we use $z = b^i$ to distinguish a^i and a^j?).
e. $L = \{x \in \{a, b\}^* \mid n_a(x) < 2n_b(x)\}$ consisting of strings with less than twice as many a's as b's.

Let a^i and a^j be two different elements of $\{a^n \mid n \geq 0\}$. Without loss of generality, assume that $i < j$. We now give two different solutions, which are both correct:

- The idea is to let z consist of only b's: just enough to make a^iz an element of L. This number of b's should, on the other hand, be too small to make a^jz an element of L.

  Extension z must contain $\lfloor \frac{i}{2} \rfloor + 1$ b's to make a^iz an element of L. One can verify that this works well both if i is even and if i is odd. Can we be certain that with this z, the string a^jz is not an element of L? To answer this question we compare $n_b(a^jz)$ to $n_a(a^jz)$.

  Let us first assume that i is odd. Then

$$\begin{aligned} 2n_b(a^jz) &= 2(\lfloor \frac{i}{2} \rfloor + 1) \\ &= 2(\frac{i-1}{2} + 1) \\ &= i - 1 + 2 = i + 1 \leq j = n_a(a^jz) \end{aligned}$$

  The last inequality holds because $i < j$. Indeed, in this case, a^jz is not an element of L.

  Now, let us assume that i is even. Then

$$\begin{aligned} 2n_b(a^jz) &= 2(\lfloor \frac{i}{2} \rfloor + 1) \\ &= 2(\frac{i}{2} + 1) \\ &= i + 2 \end{aligned}$$

  If $j = i + 1$, then this final number $i + 2$ is larger than $j = n_a(a^jz)$. In that case, a^jz is also an element of L, and z does not distinguish a^i and a^j with respect to L. For example, if $i = 8$ and $j = 9$, then $z = b^{4+1} = b^5$ does not distinguish a^i and a^j.

Hence, if i is even, the proposed string z does not suit our needs. A small adjustment is, however, suffcient. If i is even (and only then), then we may simply prepend an a to z, thus making the number of a's in $a^i z$ odd, after all: $z = a \cdot b^{\lfloor \frac{i}{2} \rfloor + 1}$. One can verify that this string z does distinguish a^i and a^j if i is even.

- A simpler solution, but maybe harder to think of. The string $z = a^j b^j$, containing equally many a's and b's, works for all i and j with $i < j$. Indeed,

$$\begin{aligned} n_a(a^i z) &= i + j < 2j = 2n_b(a^i z) \\ n_a(a^j z) &= j + j = 2j = 2n_b(a^j z) \end{aligned}$$

Hence, $a^i z \in L$, whereas $a^j z \notin L$.

f. $L = \{x \in \{a,b\}^* \mid$ no prefix of x has more b's than a's $\}$; the strings a^i and a^j with $0 \leq i < j$ are distinguished by b^j, because $a^i b^j \notin L$ and $a^j b^j \in L$.

2.22

d. Hint: note that $j = i$ is also a multiple of i.

e. $L = \{x \in \{a,b\}^* \mid n_a(x) < 2n_b(x)\}$ is not regular, which is proved using the pumping lemma (Theorem 2.29).

Suppose that L can be accepted by a finite automaton M. Let $n \geq 1$ be the number of states of M.

We choose $x = a^{2n-1}b^n$. Indeed $|x| = 3n - 1 \geq n$ and $n_a(x) = 2n - 1 < 2n = 2n_b(x)$, so $x \in L$ (note that there are many other strings x that could be used to find a contradiction).

Let u, v, w be arbitrary strings in $\{a,b\}^*$ such that $x = a^{2n-1}b^n = uvw$, $|uv| \leq n$ and $|v| \geq 1$. Then obviously, uv consists of only a's. In particular, so does v, say $v = a^k$ for some k with $1 \leq k \leq n$.

Now consider the string $x' = uv^2w$ (hence, we choose $m = 2$). We have $x' = uv^2w = uvvw = a^{2n-1+k}b^n$. As $n_a(x') = 2n - 1 + k \geq 2n - 1 + 1 = 2n = 2n_b(x')$, x' does not satisfy $n_a(x') < 2n_b(x')$, and so $x' \notin L$, regardless of what u, v and w look like exactly.

This contradicts the pumping lemma for regular languages. Therefore, the assumption that L can be accepted by a finite automaton must be wrong. We conclude that L cannot be accepted by a finite automaton.

h. $L = \{ss \mid s \in \{a,b\}^*\}$ is not regular, which is proved using the pumping lemma (Theorem 2.29).

Suppose that L can be accepted by a finite automaton M. Let $n \geq 1$ be the number of states of M.

We choose $x = a^n b^n a^n b^n$. Indeed $|x| = 4n \geq n$ and $x \in L$.

Let u, v, w be arbitrary strings in $\{a, b\}^*$ such that $x = a^n b^n a^n b^n = uvw$, $|uv| \leq n$ and $|v| \geq 1$. Then obviously, uv consists of only a's. In particular, so does v, say $v = a^k$ for some k with $1 \leq k \leq n$, in the first block of a's in x.

Now consider the string $x' = uv^2w$ (hence, we choose $m = 2$). We have $x' = uv^2w = uvvw = a^{n+k}b^n a^n b^n$, which should be a string in L according to the pumping lemma. Thus the length of this string (and hence k) must be even. Clearly, its second half then starts in its first block of b's (at position $2n + k/2 + 1$). Since x' itself begins with an a, it cannot be of the form ss, and thus cannot be in L, regardless of what u, v and w look like exactly.

This contradicts the pumping lemma for regular languages. Therefore, the assumption that L can be accepted by a finite automaton must be wrong. We conclude that L cannot be accepted by a finite automaton.

• *Use the pumping lemma for regular languages to show that the following language is not regular:* $L = \{st \mid s, t \in \{a, b\}^* \wedge (t = s \vee t = s^r)\}$
The proof proceeds along the same lines as that for $\{ss \mid s \in \{a, b\}^*\}$ (see Exercise 2.22(h)). We also choose the same string $x = a^n b^n a^n b^n$. In the end, after observing that $x' = uvvw$ cannot be of the form ss, we also observe that x' cannot be of the form ss^r, because the string starts with a and ends with b. Hence, x' cannot be in L.

2.24 The proof of this generalization of the pumping lemma is very similar to that of the pumping lemma itself.

Assume that language L is accepted by an FA M. Let n be the number of states of M.

Now, let x be an arbitrary element of L satisfying $|x| \geq n$, and let $x_1x_2x_3$ be an arbitrary decomposition of x (hence, $x = x_1x_2x_3$) such that $|x_2| = n$.

In processing input x, M first reads the letters of x_1, then those of x_2, and then those of x_3. While reading the n letters of x_2, M moves from state to state. Doing so, it 'sees' $n + 1$ states, from just before the first letter of x_2 until just after the last of x_2. As M only has n different states, it must see at least one state q at least two times while reading the letters of x_2.

Let u be the prefix of x_2 after which M is in state q for the first time (after having read x_1), and let u' be the prefix of x_2 after which M is in state q for the second time. Obviously, u is a prefix of u'. Hence, $u' = uv$ for

some string v satisfying $|v| \geq 1$. Let w be the final part of x_2, after prefix $u' = uv$: $x_2 = uvw$.

When M processes input $x = x_1x_2x_3 = x_1uvwx_3$, it traverses a cycle from state q just before substring v to state q just after substring v. M might skip this cycle (corresponding to input $x_1uv^0wx_3$) or traverse this cycle multiple times (input $x_1uv^mwx_3$ for some $m \geq 2$). In all cases M ends up in the same state as it does for input $x = x_1uv^1wx_3$, which is an accepting state.

We conclude that for every $m \geq 0$, M accepts $x_1uv^mwx_3$, i.e., $x_1uv^mwx_3 \in L$. □

2.27

a. Let M_1 en M_2 be two FA's over an input alphabet Σ.
Are there strings in Σ^* that are accepted by neither of the two FA's is the same as saying: Is $\Sigma^* - (L(M_1) \cup L(M_2)) \neq \emptyset$?
Algorithm: First construct an FA M_3 with $L(M_3) = L(M_1) \cup L(M_2)$ (see for example the product construction in the proof of Theorem 2.15). Then construct an FA M_4 for the complement $\Sigma^* - L(M_3)$ (reversing accepting states and nonaccepting states). Finally, use the algorithm from Example 1.21 or the algorithm from page 67 to decide whether or not $L(M_4)$ is empty.
c. Given an FA M accepting a language $L \subseteq \Sigma$, and given two strings $x, y \in \Sigma^*$. Are x and y distinghuishable with respect to L?
Algorithm: Minimize M as described in Section 2.6. The resulting FA $M_1 = (Q, \Sigma, q_0, A, \delta)$ still accepts L. Follow in M_1 from q_0 the path labelled by x and also the path labelled by y and check if they lead to the same state. If so, then x and y are not distinguishable with respect to L; if not, then x and y *are* distinguishable with respect to L.
(Note that as minimal FA, M_1 has the property that $\delta^*(q_0, x) = \delta^*(q_0, y)$ if and only if $x \; I_L \; y$).

2.28

The answer is no. We give an example of a language $L \subseteq \{a, b\}^*$ and an integer k such that each string $x \in \{a, b\}^*$ can be extended with a string z with $|z| \leq k$ to a string $xz \in L$.
Consider the complement of Pal: $L = \{a, b\}^* - Pal = \{w \in \{a, b\}^* \mid w \neq w^r\}$. This language is not regular, because Pal is not regular. (The complement of a regular language is always regular.)
We set $k = 2$ and argue that every string x over $\{a, b\}$ can be extended to a string that is not a palindrome by adding at most 2 letters.
if $x = \Lambda$, then $\Lambda ab = ab \in L$;

if $x = ay$ for some $y \in \{a, b\}^*$, then $xb = ayb \in L$ and
if $x = by$ for some $y \in \{a, b\}^*$, then $xa = bya \in L$.

2.29
a. False: $pal \subseteq \{a, b\}^*$ is a nonregular subset of the regular language $\{a, b\}^*$.
b. False: Nonregular languages have finite subsets, and every finite language is regular.
c. False: The union of any language over an alphabet Σ and its complement is Σ^* which is regular.
d. False: The intersection of any language and its complement is the empty language which is regular.
e. True: The complement of a regular language is always regular (Theorem 3.4). Consequently a language is regular if and only if its complement is regular.
f.: false;
g.: true;
h., **i.**: both false.
j. False: Let for all $i \geq 1$, $L_i = \{a, b\}^* - K_i$ with $K_i = \{a^{2^{i \times n}} b^{2^{i \times n}} \mid n \geq 0\}$.
Thus $K_1 = \{ab, a^2b^2, a^4b^4, a^8b^8, a^{16}b^{16}, \ldots\}$, $K_2 = \{ab, a^4b^4, a^{16}b^{16}, \ldots\}$, $K_3 = \{ab, a^8b^8, a^{64}b^{64}, \ldots\}$ etc.
Clearly, $K_{i+1} \subseteq K_i$ for all $i \geq 1$ which implies that $L_i \subseteq L_{i+1}$ for all $i \geq 1$.
Furthermore it is not too difficult to see that each K_i is not regular, and hence each L_i is not regular.
Note that the shortest string in each K_i is ab and the next shortest string is $a^{2^i} b^{2^i}$. Consequently, $\bigcap_{i=1}^{\infty} K_i = \{ab\}$, a regular language. This implies that $\bigcup_{i=1}^{\infty} L_i = \{a, b\}^* - \bigcap_{i=1}^{\infty} K_i = \{a, b\}^* - \{ab\}$ is also a regular language.

2.32 If $L \subseteq \{a, b\}^*$ consists of one single equivalence class w.r.t. I_L, then are all strings in $\{a, b\}^*$ equivalent. Therefore, for every $x, y \in \{a, b\}^*$ it holds that $x = x\Lambda \in L$ if and only if $y = y\Lambda \in L$.
Thus $L = \emptyset$ or $L = \{a, b\}^*$.

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

• *Find a language $L \subseteq \{a, b\}^*$ such that every equivalence class of I_L has exactly one element.*
The language *Pal* consisting of all palindromes over $\{a, b\}$ is an example (see Example 2.27).

2.34 Let L be a language over an alphabet Σ and let $x \in \Sigma^*$ be such that x is *not* a prefix of a string in L. Then for all strings xy with $y \in \Sigma^*$, there is no z such that $xyz \in L$. Consequently, the set $S = \{w \in \Sigma^* \mid$ there exists no z such that $wz \in L\}$ is infinite.
We need to prove that S is an equivalence class of I_L: let u, v be two strings from S. Then for all strings $z \in \Sigma^*$ we have that $uz \notin L$ and $vz \notin L$ since neither u nor v is a prefix of a string in L. Thus, $u \; I_L \; v$. Hence all strings in S are indistinguishable with respect to L and so S is contained in a single equivalence class of I_L.
Next let $u \in S$ and $v \notin S$. Then by definition of S, there is a string z such that $vz \in L$, but $uz \notin L$. Hence u and v are distinguishable with respect to L and thus belong to different equivalence classes. Together with the above this implies that S is exactly one infinite equivalence class of I_L.

• *Show that if $L \subseteq \Sigma^*$ is a language, $x \in \Sigma^*$, and $[x]$ (the equivalence class of I_L containing x) is finite, then x is a prefix of an element of L.*
Let L be a language over an alphabet Σ. Let $x \in \Sigma^*$ be a string such that its equivalence class $[x]$ is finite.
Assume that x is *not* a prefix of a string in L. As we have seen in Exercise 2.34, the set $S = \{w \in \Sigma^* \mid$ there exists no z such that $wz \in L\}$ is an infinite equivalence class of I_L. Obviously, $x \in S$, and thus $[x] = S$. This, however, contradicts the fact that $[x]$ is finite. Thus our assumption was wrong and it must be the case that x is a prefix of a string in L.

2.35 Let $L \in \Sigma^*$. If the equivalence class $[\Lambda]$ of I_L contains a string w different from Λ, then for all $z \in \Sigma^*$ we have:
$wz \in L$ if and only if $\Lambda z = z \in L$.
It follows that for all $i \geq 0$ and for all $y \in \Sigma^*$:
$w^{i+1}y = w(w^i y) \in L$ if and only if $w^i y = \Lambda(w^i y) \in L$.
Thus $w^{i+1} \; I_L \; w^i$ for all $i \geq 0$, that is, $[\Lambda]$ contains $\{w\}^*$ and hence is infinite (because $w \neq \Lambda$).

2.36 $L \subseteq \{a, b\}^*$ is a language for which we are asked to give a finite automaton. We apply the construction used to prove Theorem 5.1.
I_L has 4 equivalence classes $[\Lambda]$, $[a]$, $[ab]$, and $[b]$ which we will use as states. $[\Lambda]$ will be the initial state. Moreover, since $ab \in L$ and $\Lambda, a, b \notin L$, we designate $[ab]$ as the only accepting 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 $a \; I_L \; aa$, 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 string in L. Hence (using Exercise 2.34), $[b]$ is the equivalence class consisting of all strings which can never be extended to a string in L. In the automaton this equivalence class is a sink: $\delta([b], a) = [ba] = [b]$ and $\delta([b], b) = [bb] = [b]$.
(Draw the automaton.)

• *Suppose there is a 3-state FA accepting $L \subseteq \{a, b\}^*$. Suppose $\Lambda \notin L$, $b \notin L$, and $ba \in L$. Suppose also that aI_Lb, ΛI_Lbab, aI_Laaa, and bI_Lbb. Draw an FA accepting L.*
$L \subseteq \{a, b\}^*$ is a language that can be accepted by a 3-state FA. We use the construction in the proof of Theorem 2.36 to specify this automaton on basis of the given data.
We let $[\Lambda]$ be its initial state. $[\Lambda]$ is not an accepting state, because $\Lambda \notin L$.
$\delta([\Lambda], a) = [a] = [b] = \delta([\Lambda], b)$, because $a\ I_L\ b$. Since $ba \in L$ and $a \notin L$, the state $[a] = [b]$ is different from $[\Lambda]$. Moreover $[b]$ is not accepting, because $b \notin L$.
$\delta([b], a) = [ba]$ and $\delta([b], b) = [bb] = [b]$. We know that $ba \in L$ and so $[ba]$ is an accepting state and consequently differs from $[\Lambda]$ and $[b]$.
$\delta([ba], a) = [baa] = [aaa]$ because $[b] = [a]$; since $[aaa] = [a]$ and $[a] = [b]$, it follows that $\delta([ba], a) = [b]$; $\delta([ba], b) = [bab] = [\Lambda]$.
(Now the FA can be drawn; by the way, in our argumentation, we did not need the a priori given fact that the FA has 3 states.)

• *Suppose there is a 3-state FA accepting $L \subseteq \{a, b\}^*$. Suppose $\Lambda \notin L$, $b \in L$, $ba \notin L$, and $baba \in L$, and that ΛI_La and aI_Lbb. Draw an FA accepting L.*
$L \subseteq \{a, b\}^*$ is accepted by an FA with 3 states. Using the data given we can identify such an FA.
Its initial state is $[\Lambda]$ which is not an accepting state because $\Lambda \notin L$. Since $\Lambda\ I_L\ a$, we have $\delta([\Lambda], a) = [a] = [\Lambda]$; $\delta([\Lambda], b) = [b]$ and because $b \in L$, state $[b]$ is accepting and hence different from $[\Lambda]$.
$\delta([b], a) = [ba]$ and because $ba \notin L$, this is not an accepting state and hence different from $[b]$. If $[ba] = [\Lambda]$ would hold, then $\delta([ba], b) = \delta([\Lambda], b] = [b]$ and with $\delta([b], a] = [ba]$ this implies that reading $baba$ from $[\Lambda]$ would lead to $[ba] = [\Lambda]$ which is a non-accepting state. This is in contradiction with $baba \in L$. Thus $[ba]$ cannot be $[\Lambda]$, but is a new, third state.
$\delta([b], b) = [bb] = [a] = [\Lambda]$ because $a\ I_L\ bb$.
$\delta([ba], a) = [baa]$ and $\delta([ba], b) = [bab]$. Because we follow the construction for a minimal automaton and 3 states should be sufficient, we have to identify

$[baa]$ and $[bab]$ with one of the states $[\Lambda]$, $[b]$, and $[ba]$.
We use again the fact that $baba \in L$. Thus $\delta([bab], a) = [b]$, the only accepting state. Since $\delta([\Lambda], a) \neq [b]$ and $\delta([b], a] \neq [b]$, it must be the case that $[bab] = [ba]$. We can now also conclude that $[baa] = \delta([ba], a) = \delta([bab], a) = [baba] = [b]$, because $baba \in L$ and $[b]$ is the only accepting state.
Draw this FA.

2.38
a. Find all languages $L \subseteq \{a, b\}^*$ for which I_L has the following three equivalence classes: $E_1 = \{a, b\}^*\{b\}$, $E_2 = \{a, b\}^*\{ba\}$, and $E_3 = \{a, b\}^* - (E_1 \cup E_2)$.
These languages share the property that they are accepted by a minimal FA with 3 states corresponding to E_1, E_2, E_3. The initial state is always E_3 since $\Lambda \in E_3$, that is $E_3 = [\Lambda]$. Of the transition functions δ we know that always:
$\delta(E_3, a) = [a] = E_3$ and $\delta(E_3, b) = [b] = E_1$;
$\delta(E_1, a) = [ba] = E_2$ and $\delta(E_1, b) = [bb] = E_1$;
$\delta(E_2, a) = [baa] = E_3$ and $\delta(E_2, b) = [bab] = E_1$.
Consequently, the only freedom in the definition of this FA is the choice of a set of accepting states. There are 8 possibilities:
$A_1 = \emptyset$, then $L_1 = \emptyset$,
$A_2 = \{E_1\}$, then $L_2 = E_1$,
$A_3 = \{E_2\}$, then $L_3 = E_2$,
$A_4 = \{E_3\}$, then $L_4 = E_3$,
$A_5 = \{E_1, E_2\}$, then $L_5 = E_1 \cup E_2$,
$A_6 = \{E_1, E_3\}$, then $L_6 = E_1 \cup E_3$,
$A_7 = \{E_2, E_3\}$, then $L_7 = E_2 \cup E_3$, and
$A_8 = \{E_1, E_2, E_3\}$, then $L_8 = \{a, b\}^*$.
Some of these languages however define less than three equivalence classes: $L_1 = \emptyset$ and $L_8 = \{a, b\}^*$ both define only one equivalence class: $E_1 \cup E_2 \cup E_3 = \{a, b\}^*$. Also $L_2 = E_1 = \{a, b\}^*\{b\}$ and its complement $L_7 = E_2 \cup E_3$ can be defined by an FA with fewer (two) states (corresponding to E_1, strings ending with b, and $E_2 \cup E_3$, strings not ending with b).
For the remaining four languages, L_3, L_4, L_5, L_6, the given automaton is indeed minimal and hence these are the languages asked for. When you draw the FA it is clear that now no states can be identified. In particular E_3 is accepting if and only if E_2 is not accepting.

b. Find all languages $L \subseteq \{a, b\}^*$ for which I_L has the following three equiva-

lence classes: $(\{a,b\}\{a\}^*b)^*$ and $(\{a,b\}\{a\}^*b)^*\{a\}\{a\}^*$ and $(\{a,b\}\{a\}^*b)^*\{b\}\{a\}^*$ (in the book, this third equivalence class starts with $\{a,b\}^*$, but that is incorrect).
Note that indeed these sets have no strings in common and together form $\{a,b\}^*$.
This means that we can make eight different languages by combining all languages that are either in or not in the individual equivalence classes. This way we get eight languages.
However, not every combination will yield a language with *three* equivalence classes, as required by the exercise. An example is the empty set $\emptyset$ obtained by the empty combination (all strings are indistinguishable with respect to $\emptyset$ which thus defines only one equivalence class). Another example is its complement $\{a,b\}^*$ obtained by taking the union of all three sets (also $\{a,b\}^*$ has only one associated equivalence class).
Also the language consisting of the strings from the first equivalence class, that is the language $L_1 = (\{a,b\}\{a\}^*\{b\})^*$ does not define three equivalence classes, but only two. (This can be seen by constructing a two-state FA for L_1.) Consequently also the complement of L_1 which is the union of the two other equivalence classes defines only two equivalence classes.
Thus we are left with the four languages: $L_2 = (\{a,b\}\{a\}^*b)^*\{a\}\{a\}^*$, $L_3 = (\{a,b\}\{a\}^*b)^*\{b\}\{a\}^*$, $L_1 \cup L_2$, and $L_1 \cup L_3$. Each of these languages defines three equivalence classes:
Prove that both L_2 and L_3 define three equivalence classes (show that three states are enough to recognize them and give 3 different strings that are not equivalent to one another). Then note that L_2 and $L_1 \cup L_3$ are complements and also L_3 and $L_1 \cup L_2$ are complements of one another.

2.39 $L = \{a^n b^n \mid n \geq 0\}$. (See also Example 2.37 and Exercise 2.17)
The partition of $\{a,b\}^*$ in equivalence classes of I_L is the same as for the language $L - \{\Lambda\}$, namely: $\{a^i\}$, for each $i \geq 0$, $\{a^i b^j\}$ for all $i > j > 0$, and $\{x \in \{a,b\}^* \mid \nexists z \text{ such that } xz \in L\}$.

2.40 $L = \{x \in \{a,b\}^* \mid n_a(x) = n_n(x)\}$, the language consisting of all strings over $\{a,b\}$ with an equal number of a's and b's.
a. For any two strings $x, y \in \{a,b\}^*$ it is the case that $x \; I_L \; y$ whenever the difference between the number of a's and b's in x and y is the same. To prove this statement assume that $n_a(x) - n_b(x) = n_a(y) - n_b(y)$ and consider an arbitrary string $z \in \{a,b\}^*$.
Then $xz \in L$ if and only if
$n_a(xz) - n_b(xz) = 0$ if and only if

$n_a(x) + n_a(z) - (n_b(x) + n_b(z)) = n_a(x) - n_b(x) + n_a(z) - n_b(z) = 0$
if and only if
$n_a(y) - n_b(y) + n_a(z) - n_b(z) = 0$ if and only if
$n_a(yz) - n_b(yz) = 0$ if and only if $yz \in L$.
b. Conversely, for any two strings $x, y \in \{a, b\}^*$ it is the case that if $x \; I_L \; y$, then the difference between the number of a's and b's in x and y is the same. To prove this statement we consider two strings $x, y \in \{a, b\}^*$ such that $n_a(x) - n_b(x) \neq n_a(y) - n_b(y)$. Now let $z \in \{a, b\}^*$ be such that $n_b(z) - n_a(z) = n_a(x) - n_b(x)$. Consequently, $n_a(xz) - n_b(xz) = n_a(x) + n_a(z) - n_b(x) - n_b(z) = (n_a(x) - n_b(x)) - (n_b(z) - n_a(z)) = 0$. Thus $xz \in L$. However, since $n_a(y) - n_b(y) \neq n_a(x) - n_b(x)$, we know that $n_a(y) - n_b(y) \neq n_b(z) - n_a(z)$ and thus $n_a(yz) - n_b(yz) = (n_a(y) - n_b(y)) - (n_b(z) - n_a(z)) \neq 0$ which implies that $yz \notin L$. Hence $x \; I_L \; y$ does not hold. Summarizing: if $x \; I_L \; y$ it must be the case that $n_a(x) - n_b(x) = n_a(y) - n_b(y)$.
c. From **a.** and **b.** it follows that an equivalence class of I_L containing a string x is determined by the difference $n_a(x) - n_b(x)$: a string y is in $[x]$ if and only if $n_a(y) - n_b(y) = n_a(x) - n_b(x)$.
This means that we have the following equivalence classes:
$\ldots, [b^k], \ldots, [b^3], [b^2], [b], [\Lambda], [a], [a^2], [a^3], \ldots, [a^k], \ldots,$
where for all $k \geq 0$, the equivalence class $[b^k]$ consists of all strings w over $\{a, b\}$ with $n_b(w) - n_a(w) = k$, and the equivalence class $[a^k]$ consists of all strings w over $\{a, b\}$ with $n_a(w) - n_b(w) = k$.
Note that I_L has an infinite number of equivalence classes which according to Theorem 2.36 implies that L is not a regular language.

2.55 See Figure 2.45.
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	=

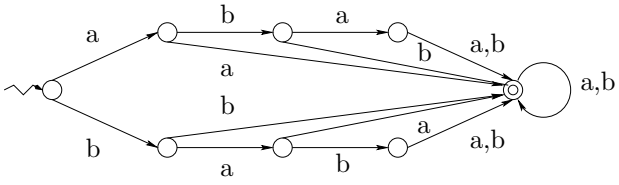
Consequently, we have the following equivalence classes: $\{1,3\}$, $\{2,4\}$ and $\{5\}$, which gives a minimal FA with three states $q_0 = \{1,3\}$, the initial state; $q_1 = \{2,4\}$; and $q_3 = \{5\}$, the only accepting state. Its transition function is given in the next table:

	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 FA.
b. The given FA is already minimal.

2.57 You are asked for a number of languages over the alphabet $\{a,b\}$ to determine whether they can be accepted by an FA or not. If you do well (go after the number of equivalence classes, use the pumping lemma or try to make up an FA for the language), then you will find that only the language given in **b.** is regular.
a. Let $L = \{x \in \{a,b\}^* \mid \exists w,y \in \{a,b\}^*.w \neq \Lambda \wedge x = wwy\}$, and take two strings $v = ab^n$ 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 strings ab^n for $n > 0$ are pairwise distinguishable, and therefore there can be no FA recognizing L, as it would need infinitely many states by Theorem 2.26.
b. Let $L = \{x \in \{a,b\}^* \mid \exists y,w,z : w \neq \Lambda \wedge x = ywwz\}$.
Claim: $x \in L$ if and only if x contains aa, bb, $abab$ or $baba$ as a substring.
Proof of the claim: if aa, bb, $abab$ of $baba$ a substring is of x, dan is x by definition in L.
Conversely, assume that $x \in L$. Then $|x| \geq 2$. Let aa and bb be not substrings 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 substring. Analogously, if $w = bau$, then must be $u = b$ and is $x = ywwz = ybabbabz$, contradicting our assumption that x does not contain bb as substring.
Summarizing, if $x \in L$, then x contains at least one of the strings aa, bb, $abab$ and $baba$ as substring.
Now it is easy to draw an FA that recognizes L:



2.58 Consider $L = AEqB = \{x \in \{a,b\}^* : n_a(x) = n_b(x) \geq 0\}$, the languague consisting of all strings with as many a's as b's. According to Exercise 2.40 and Example 2.30, this language is not regular. Now $L^* = L$, because every concatenation of (equal or different) strings from L yields another string with as many a's as b's. Thus, L^* is not regular.

2.59 Consider $L = Pal$, the language of all palindromes over $\{a,b\}$. We know that Pal is not regular (see Example 2.27). On the other hand, the strings Λ, a and b are elements of Pal. Consequently, every string over $\{a,b\}$ is a concatenation of palindromes. This implies that $Pal^* = \{a,b\}^*$, which is a regular language.

---

version of 4 September 2024, feel free to mention any errors in these solutions at rvvliet@liacs.nl