Chapter 2

Finite Automata and Regular Languages

In this chapter we introduce the notion of a deterministic finite automaton, of a non-deterministic finite automaton with silent steps and of a regular expression. We will show that the class of associated languages, the class of regular languages, is the same for all these three concepts. We study closure properties of the class of regular languages and provide a means to prove that a language is not regular.

2.1 Deterministic finite automata

We start off with the simplest yet most rigid concept of the three main notions mentioned.

Definition 2.1 (Deterministic finite automaton)**.** A deterministic finite automaton (DFA) is a tuple $D = (Q, \Sigma, \delta, q_0, F)$ with Q a finite non-empty set, the set of states, Σ a finite set, the alphabet, $\delta : Q \times \Sigma \to Q$ the transition *function*, $q_0 \in Q$ the initial state, and $F \subseteq Q$ the set of final states.

We sometimes write $q \xrightarrow{a}_D q'$ instead of $\delta(q, a) = q'$, and call it a transition of D from state q to state q' on input or symbol a. We may write $q \xrightarrow{a} q'$ if the automaton D is clear from the context. Intuitively, when automaton D is in state q and the symbol a is the first symbol on input, the automaton D moves to state q' while consuming the symbol a. Unlike for non-deterministic finite automata or NFA that we encounter in the

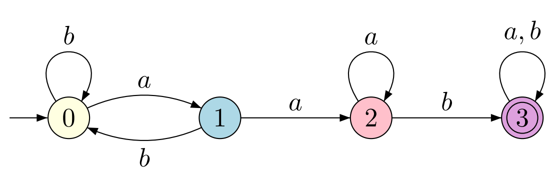


Figure 2.1: Finite automaton of Example 2.2

next section, for a DFA in each state $q \in Q$ and for every symbol $a \in \Sigma$ the next state, which is the state $\delta(q,a)$, is determined by the transition function δ.

Example 2.2. Figure 2.1 gives a visual representation of a deterministic finite automaton, D say. The set of states is $\{q_0, q_1, q_2, q_3\}$ with q_0 the initial state as indicated by the small incoming arrow. The alphabet Σ consists of the symbols a and b. The δ-function is indicated by the arrows between states. E.g., for q_0 there is an arrow labeled a to q_1, thus $\delta(q_0,a) = q_1$. There is also an arrow labeled b from q_0 to itself, so $\delta(q_0,b) = q_0$. The self-loop of q_3 labeled a, b represents two transitions, one for a and one for b. Thus $\delta(q_3,a) = q_3$ and $\delta(q_3,b) = q_3$. There is one final state, viz. q_3, as indicated by the double boundary of the state. Thus, the set of final states is $\{q_3\}$.

Formally, $D = (Q, \Sigma, \delta, q_0, F)$ where $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{a, b\}$, $F = \{q_3\}$ and $\delta \colon Q \times \Sigma \to Q$ is given by the table below.

	a	b
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_3
q_3	q_3	q_3

Note, since $\delta : Q \times \Sigma \to Q$ is a function, for each state $q \in Q$ there is exactly one state, viz. the state $\delta(q,a) \in Q$, for each symbol $a \in \Sigma$.

A configuration of a finite automaton $D = (Q, \Sigma, \delta, q_0, F)$ is a pair (q,w) of a state $q \in Q$ and a string $w \in \Sigma^*$. The configuration (q,w) indicates that D is in state q with the word w on input. We write $(q,w) \vdash_D (q',w')$ if automaton D in state q moves to state q' when reading the first, i.e. leftmost, symbol of w. More specifically, the relation $\vdash_D \subseteq (Q \times \Sigma^*) \times (Q \times \Sigma^*)$ is defined by

$$(q,w) \vdash_D (q',w') \quad \text{iff} \quad w = aw' \text{ and } \delta(q,a) = q', \text{ for some } a \in \Sigma$$

We say that (q,aw') yields (q',w') with respect to D, or that D derives configuration (q',w') from configuration (q,aw') in one step. By definition, for each $q \in Q$ there exist no $q' \in Q$ and $w' \in \Sigma^*$ such that $(q,\varepsilon) \vdash_D (q',w')$.

Note $\vdash_D \subseteq (Q \times \Sigma^*) \times (Q \times \Sigma^*)$ is a relation on $Q \times \Sigma^*$. We denote by $\vdash_D^*$ the reflexive and transitive closure of $\vdash_D$. Thus

$$\begin{aligned}
&(q,w) \vdash_D^* (q',w') \quad \text{iff} \\
&\qquad \exists n \geqslant 0\, \exists w_0,\ldots,w_n \in \Sigma^*\, \exists q_0,\ldots,q_n \in Q\colon \\
&\qquad\qquad (q,w) = (q_0,w_0),\ (q_{i-1},w_{i-1}) \vdash_D (q_i,w_i),\ \text{for } 1 \leqslant i \leqslant n, \\
&\qquad\qquad\quad \text{and } (q_n,w_n) = (q',w')
\end{aligned}$$

In the above situation we say that (q,w) yields (q',w') with respect to D, or that D derives configuration (q',w') from configuration (q,w) in a number –zero, one, or more– steps.

Example 2.3. Continuing Example 2.2 involving the DFA D given by Figure 2.1, we have for the relation $\vdash_D$ that $(q_1, aaaba) \vdash_D (q_2, aaba)$, $(q_2, aaba) \vdash_D (q_2, aba)$, $(q_2, aba) \vdash_D (q_2, ba)$, $(q_2, ba) \vdash_D (q_3, a)$, and, $(q_3, a) \vdash_D (q_3, \varepsilon)$. For the relation $\vdash_D^*$ we have $(q_1, aaaba) \vdash_D^* (q_3, \varepsilon)$, but also $(q_1, aaaba) \vdash_D^* (q_1, aaaba)$, $(q_2, aaba) \vdash_D^* (q_2, aba)$, $(q_2, aaba) \vdash_D^* (q_3, \varepsilon)$, and $(q_2, aaba) \vdash_D^* (q_3, a)$.

To facilitate inductive reasoning it is technically advantageous to have a slightly more precise formulation of the 'derives' relation $\vdash_D^*$. Assume $D = (Q, \Sigma, \delta, q_0, F)$. We define, for $n \geqslant 0$, the relation $\vdash_D^n \subseteq (Q \times \Sigma^*) \times (Q \times \Sigma^*)$ as follows: For $q, q' \in Q$, $w, w' \in \Sigma^*$ we put $(q, w) \vdash_D^0 (q', w')$ iff $q = q'$ and $w = w'$, and $(q, w) \vdash_D^{n+1} (q', w')$ iff $(q, w) \vdash_D^n (\bar{q}, \bar{w})$ and $(\bar{q}, \bar{w}) \vdash_D (q', w')$ for some state $\bar{q} \in Q$ and string $\bar{w} \in \Sigma^*$. If $(q, w) \vdash_D^n (q', w')$ then there are n input symbols processed. So we expected the following property to hold.

Lemma 2.4. Let $D = (Q, \Sigma, \delta, q_0, F)$ be a DFA. For all $q, q' \in Q$, $w, w' \in \Sigma^*$ it holds that

$$(q, w) \vdash_D^* (q', w') \iff (q, w) \vdash_D^n (q', w') \text{ for } n = |w| - |w'|$$

Proof. ($\Rightarrow$) By definition of $\vdash_D^*$ we have $(q, w) \vdash_D^* (q', w')$ iff

$$(q, w) = (q_0, w_0),\ (q_{i-1}, w_{i-1}) \vdash_D (q_i, w_i),\ \text{for } 1 \leqslant i \leqslant n,\ \text{and } (q_n, w_n) = (q', w')$$

for suitable $n \geqslant 0$, $q_0, \ldots, q_n \in Q$, and $w_0, \ldots, w_n \in \Sigma^*$. By definition of $\vdash_D$ we have $w_{i-1} = a_i w_i$ for some $a_i \in \Sigma$, $1 \leqslant i \leqslant n$. Therefore,

$$(q, w) = (q_0, w_0) \vdash_D (q_1, w_1) \vdash_D \cdots \vdash_D (q_n, w_n) = (q', w')$$

and $w = w_0 = a_1 \cdots a_n w_n = a_1 \cdots a_n w'$. Thus, $|w| = n + |w'|$.

($\Leftarrow$) One can show by induction on n: if $(q, w) \vdash_D^n (q', w')$ then $(q, w) \vdash_D^* (q', w')$, the details of which are omitted here. $\square$

A special case of the lemma above is when $w' = \varepsilon$. Since $|\varepsilon| = 0$, we have $n = |w|$ and obtain $(q, w) \vdash_D^* (q', \varepsilon) \iff (q, w) \vdash_D^{|w|} (q', \varepsilon)$.

For a DFA, given a state q and an symbol a on input, the next state is determined. It is the state $\delta(q, a)$. Also, if it takes the string w to be read off from input to get from state q to some state q' while leaving a string w' on input, then extending the input with a string v doesn't influence this derivation of the DFA from q to q'. Thus, we have the following two properties for the 'derives' relation $\vdash_D^*$.

Lemma 2.5. Let $D = (Q, \Sigma, \delta, q_0, F)$ be a DFA.

(a) For all states $q, q', q'' \in Q$ and words $w, w' \in \Sigma^*$ it holds that

$$(q, w) \vdash_D^* (q', w') \wedge (q, w) \vdash_D^* (q'', w') \implies q' = q''$$

(b) For states $q, q' \in Q$ and all words $w, w', v \in \Sigma^*$ it holds that

$$(q, w) \vdash_D^* (q', w') \iff (q, wv) \vdash_D^* (q', w'v)$$

Proof. For the proof we first prove a stronger property. *Claim*: for all $q, q' \in Q$ and $w, w' \in \Sigma^*$ it holds that

$$\begin{aligned} &(q, w) \vdash_D^* (q', w') \quad \text{iff} \\ &\quad \exists n \geqslant 0\, \exists q_0, \dots, q_n \in Q\, \exists a_1, \dots, a_n \in \Sigma\colon \\ &\qquad q_0 = q,\ \delta(q_{i-1}, a_i) = q_i \text{ for } 1 \leqslant i \leqslant n,\ q_n = q' \\ &\qquad\quad \text{and } w = a_1 \cdots a_n w' \end{aligned} \tag{2.1}$$

Proof of the claim. ($\Rightarrow$) If $(q, w) \vdash_D^* (q', w')$, then exist $n \geqslant 0$, $q_0, \dots, q_n \in Q$, $w_0, \dots, w_n \in \Sigma^*$ such that $(q, w) = (w_0, q_0)$, $(q_{i-1}, w_{i-1}) \vdash_D (q_i, w_i)$ for $1 \leqslant i \leqslant n$, and $(q_n, w_n) = (q', w')$. Since $(q_{i-1}, w_{i-1}) \vdash_D (q_i, w_i)$ we can pick $a_i \in \Sigma$ such that $w_{i-1} = a_i w_i$ and $\delta(q_{i-1}, a_i) = q_i$ for $1 \leqslant i \leqslant n$, by definition of $\vdash_D$. It follows that $w = a_1 \cdots a_n w'$, which proves the implication.

($\Leftarrow$) Suppose $q_0, \dots, q_n \in Q$, $a_1, \dots, a_n \in \Sigma$ are such that $q_0 = q$, $\delta(q_{i-1}, a_i) = q_i$ for $1 \leqslant i \leqslant n$, and $q_n = q'$. Put $w_i = a_{i+1} a_{i+2} \cdots a_n w'$ for $0 \leqslant i \leqslant n$. Note $w_{i-1} = a_i w_i$ for $1 \leqslant i \leqslant n$. Then $(q_0, w_0) = (q, w)$, $(q_{i-1}, w_{i-1}) = (q_{i-1}, a_i w_i) \vdash_D (w_i, q_i)$ for $1 \leqslant i \leqslant n$, since $\delta(q_{i-1}, a_i) = q_i$, and $(q_n, w_n) = (q', w')$. But this means $(q, w) \vdash_D^* (q', w')$, and proves the other implication.

As to prove item (a) of the lemma, suppose $(q, w) \vdash_D^* (q', w')$ and $(q, w) \vdash_D^* (q'', w')$. By the claim, we can find $n \geqslant 0$, $q_0, \dots, q_n \in Q$, $a_1, \dots, a_n \in \Sigma$ such that $q_0 = q$, $\delta(q_{i-1}, a_i) = q_i$ for $1 \leqslant i \leqslant n$, $q_n = q'$ and $w = a_1 \cdots a_n w'$. We can also find $m \geqslant 0$, $q'_0, \dots, q'_m \in Q$, $a'_1, \dots, a'_m \in \Sigma$ such that $q'_0 = q$, $\delta(q'_{i-1}, a'_i) = q'_i$ for $1 \leqslant i \leqslant m$, $q'_m = q''$ and $w = a'_1 \cdots a'_m w'$. Since $a_1 \cdots a_n w' = w = a'_1 \cdots a'_m w'$, we have $n = m$ and $a_i = a'_i$ for $1 \leqslant i \leqslant n$. Since $q_0 = q = q'_0$ it follows that $q_i = q'_i$ for $0 \leqslant i \leqslant n$. In particular $q' = q_n = q'_m = q''$ since $n = m$.

Item (b) follows from the claim too. If $(q, w) \vdash_D^* (q', w')$ then applying Equation (2.1) from left to right, we can pick $n \geqslant 0$, $q_0, \dots, q_n \in Q$, $a_1, \dots, a_n \in \Sigma$ such that $q_0 = q$, $\delta(q_{i-1}, a_i) = q_i$ for $1 \leqslant i \leqslant n$, $q_n = q'$ and $w = a_1 \cdots a_n w'$. Since $w = a_1 \cdots a_n w'$ implies $wv = a_1 \cdots a_n w'v$, we conclude, applying Equation (2.1) right to left, that $(q, wv) \vdash_D^* (q', w'v)$. □

The first item expresses the determinacy of the DFA D: the state q and the input string w (together with the number of symbols to be processed) determine the resulting state. The second item expresses that a computation $(q, w) \vdash_D^* (q', w')$ of a DFA is only influenced by the input that is read, i.e. the prefix u of w such that $w = uw'$.

We next introduce the important concept of the language accepted by a DFA.

Definition 2.6 (Language accepted by DFA)**.** Let $D = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton. The set $\mathcal{L}(D) \subseteq \Sigma^*$, called the language accepted by D, is defined by

$$\mathcal{L}(D) = \{\, w \in \Sigma^* \mid \exists q \in F\colon (q_0, w) \vdash_D^* (q, \varepsilon) \,\}$$

Thus, a string $w \in \Sigma^*$ is in $\mathcal{L}(D)$ when starting from the initial state q_0 the DFA D reaches a final state when all of w is processed as input.

In view of the proof of Lemma 2.5, we can reformulate the condition for a string w being included in $\mathcal{L}(D)$. Suppose $w = a_1 \cdots a_n$. Put $q'_0 = q_0$ and $q'_i = \delta(q'_{i-1}, a_i)$ for $1 \leqslant i \leqslant n$. Then it holds that

$$w \in \mathcal{L}(D) \iff q'_n \in F$$

The point is that for the DFA D, the string $w = a_1 \cdots a_n$ is accepted if a final state is reached from the start state q_0 while processing $a_1, \ldots, a_n$ successively.

Example 2.7. For a DFA D we always have $\emptyset \subseteq \mathcal{L}(D) \subseteq \Sigma^*$. Extreme cases occur when D has no (reachable) final states at all, or when each (reachable) state of D is a final state. In these situation we have $\mathcal{L}(D) = \emptyset$, and $\mathcal{L}(D) = \Sigma^*$, respectively.

For a DFA D, a state q is called reachable in D if $(q_0, w) \vdash_D^* (q, \varepsilon)$ for some string $w \in \Sigma^*$. Thus, when suitable input w is provided, D will move from the initial state q_0 to the state q when processing the string w. Note, non-reachable states do not contribute to the language of the DFA.

In most other cases than those of Example 2.7 we need to look more closely to determine $\mathcal{L}(D)$. A useful notion when analyzing a DFA is that of a path set. We say that a set $L \subseteq \Sigma^*$ is the path set of q with respect to D when L consists of all words w that bring, when starting from the initial state, the automaton to state q, notation $\mathit{pathset}_D(q)$. Formally,

$$\mathit{pathset}_D(q) = \{\, w \in \Sigma^* \mid (q_0, w) \vdash_D^* (q, \varepsilon) \,\}$$

Comparing the definition of a pathset and the definition of the language accepted by a DFA, we observe, in the context of a DFA $D = (Q, \Sigma, \delta, q_0, F)$,

$$\mathcal{L}(D) = \textstyle\bigcup_{q \in F} \mathit{pathset}_D(q)$$

Clearly, by definition, when a state $q \in Q$ is not reachable in D then $\mathit{pathset}_D(q) = \emptyset$.

Example 2.8. We claim that for the automaton D of Example 2.2, depicted in Figure 2.1, we have

$$\mathcal{L}(D) \;=\; \{\, w \in \{a,b\}^* \mid w \text{ has a substring } aab \,\}$$

To see this we make an inventory of the path sets. We have

state	path set
q_0	no substring aab, not ending in a
q_1	no substring aab, ending in a, not in aa
q_2	no substring aab, ending in aa
q_3	substring aab

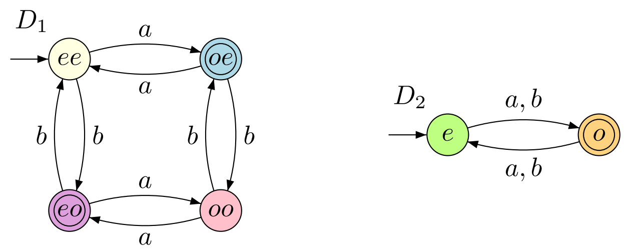


Figure 2.2: DFAs of Example 2.9

Since q_3 is the only final state of D it follows that a string $w \in \{a, b\}^*$ is accepted by D iff w has a substring aab, as claimed above.

Example 2.9. Consider the deterministic finite automaton D_1 depicted in the left part of Figure 2.2. We argue that the language $\mathcal{L}(D_1)$ of D_1 is the set

$$\mathcal{L}(D_1) = \{\, w \in \{a, b\}^* \mid \#_a(w) \text{ odd, or } \#_b(w) \text{ odd, but not both} \,\}$$

For D_1 we have the following characterization of the path sets:

state	path set
q_{ee}	$\#_a(w)$ is even, $\#_b(w)$ is even
q_{oe}	$\#_a(w)$ is odd, $\#_b(w)$ is even
q_{eo}	$\#_a(w)$ is even, $\#_b(w)$ is odd
q_{oo}	$\#_a(w)$ is odd, $\#_b(w)$ is odd

Only q_{oe} and q_{eo} are final states, thus w is accepted iff $\#_a(w)$ is odd and $\#_b(w)$ is even for reaching q_{oe}, or $\#_a(w)$ is even and $\#_b(w)$ is odd for reaching q_{eo}.

Now consider the deterministic finite automaton D_2 given by the right part of Figure 2.2. The state q_e is the initial state, the state q_o the only final state. We claim that the language $\mathcal{L}(D_2)$ of strings that are accepted by D_2 is the set of strings of odd length, i.e.

$$\mathcal{L}(D_2) = \{\, w \in \{a, b\}^* \mid |w| \text{ odd} \,\}$$

as can be seen by computing the path sets for q_e and q_o:

state	path set
q_e	$\lvert w \rvert$ even
q_o	$\lvert w \rvert$ odd

Since q_o is the only final state of D_2 the claim follows.

Notice, for arbitrary $w \in \{a, b\}^*$ we have $|w| = \#_a(w) + \#_b(w)$, and $\#_a(w) + \#_b(w)$ is odd iff $\#_a(w)$ odd and $\#_b(w)$ even, or $\#_a(w)$ even and $\#_b(w)$ odd. It follows that D_1 and D_2 accept the same language, i.e. $\mathcal{L}(D_1) = \mathcal{L}(D_2)$.

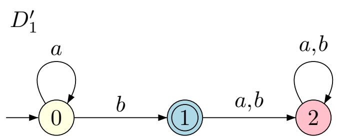
Exercises for Section 2.1

Exercise 2.1.1.

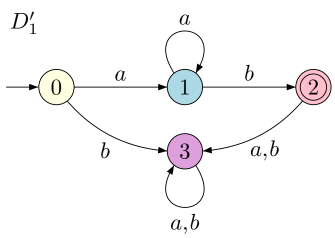
(a) Construct a DFA D_1 with alphabet $\{a,b\}$ (with no more than three states) for the language $L_1 = \{\, a^n b \mid n \geqslant 0 \,\}$ and establish with the help of path sets that $\mathcal{L}(D_1) = L_1$.

(b) Also construct a DFA D_1' over $\{a,b\}$ (now with no more than four states) for the language $L_1' = \{\, a^n b \mid n > 0 \,\}$ and again establish with the help of path sets that $\mathcal{L}(D_1') = L_1'$.

Answer to Exercise 2.1.1

(a) 

q_0	a^n	$n \geqslant 0$	
q_1	$a^n b$	$n \geqslant 0$	
q_2	$a^n b w$	$n \geqslant 0,\ w \neq \varepsilon$	string $a^n b$ followed by extra symbols

(b) 

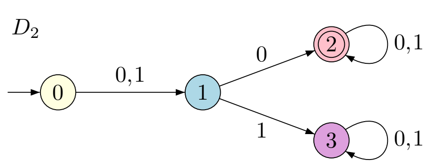
q_0	ε		forced to leave, no return
q_1	a^n	$n > 0$	
q_2	$a^n b$	$n > 0$	
q_3	$b,\ a^n b w$	$n > 0,\ w \neq \varepsilon$	string b, and strings $a^n b$ with trailing symbols

State q_2 of D_1 and state q_3 of D_1' are so-called sink states.

Exercise 2.1.2. (a) Construct a DFA D_2 with alphabet $\{0,1\}$ (with no more than four states) for the language $L_2 = \{\, w \in \{0,1\}^* \mid \text{the second element of } w \text{ is } 0 \,\}$ and establish with the help of path sets that $\mathcal{L}(D_2) = L_2$. For example, $10111 \in L_2$ while $01000 \notin L_2$.

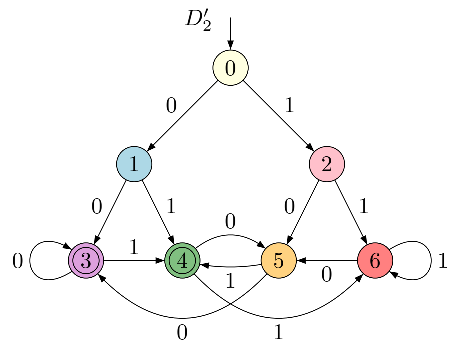
(b) Construct a DFA D_2' with alphabet $\{0,1\}$ (with no more than seven states) for the language $L_2' = \{w \in \{0,1\}^* \mid \text{the second last element of } w \text{ is } 0\}$ and establish with the help of path sets that $\mathcal{L}(D_2') = L_2'$. For example, $11101 \in L_2'$ while $00010 \notin L_2'$.

Answer to Exercise 2.1.2

(a) 

q_0	ε		forced to leave, no return
q_1	$0,\ 1$		
q_2	$00w,\ 01w$	$w \in \{0,1\}^*$	second symbol 0
q_3	$10w,\ 11w$	$w \in \{0,1\}^*$	second symbol 1

(b)



q_0	ε		
q_1	0		
q_2	1		
q_3	$w00$	$w \in \{0{,}1\}^*$	string with suffix 00
q_4	$w01$	$w \in \{0{,}1\}^*$	string with suffix 01
q_5	$w10$	$w \in \{0{,}1\}^*$	string with suffix 10
q_6	$w11$	$w \in \{0{,}1\}^*$	string with suffix 11

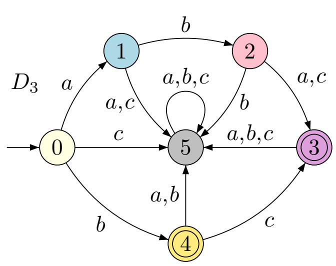
Exercise 2.1.3.

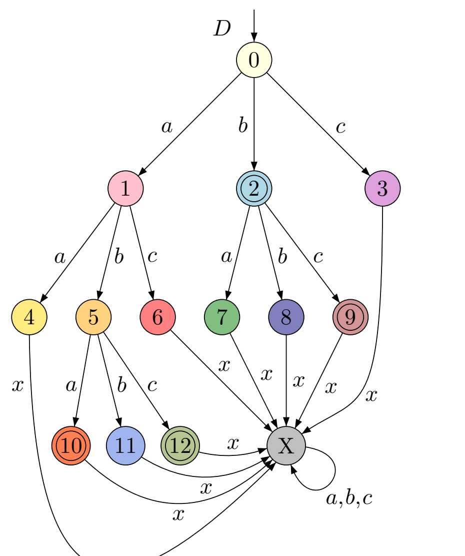
(a) Construct a DFA D_3 for the finite language $L_3 = \{aba, abc, bc, b\}$ over the alphabet $\{a, b, c\}$.

(b) If a language $L \subseteq \{a, b, c\}^*$ is finite, does there exists a DFA D such that $\mathcal{L}(D) = L$?

Answer to Exercise 2.1.3

(a)





x short for a, b, c

(b) Yes. If Σ has α symbols and the maximal length of accepted strings in L is ℓ, then a tree-like finite automaton with at most $1 + \sum_{k=1}^{\ell} \alpha^k$ can accept the finite language L. The extra summand 1 is for the sink state that catching strings longer than ℓ and, if applicable, strings of length at most ℓ not in L.

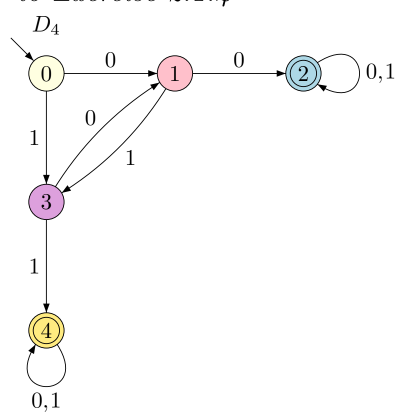
Exercise 2.1.4.

(a) Construct a DFA D_4 with alphabet $\{0,1\}$ (with no more than five states) for the language $L_4 = \{\, w \in \{0,1\}^* \mid w$ has a substring 00 or a substring 11 (or both) $\}$ and verify that $\mathcal{L}(D_4) = L_4$.

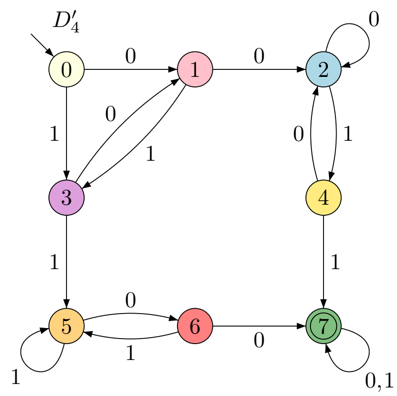
(b) Construct a DFA D'_4 with alphabet $\{0,1\}$ (with no more than eight states) for the language $L'_4 = \{\, w \in \{0,1\}^* \mid w$ has a substring 00 and a substring 11 $\}$. (No verification with path sets asked.)

Answer to Exercise 2.1.4

(a)



(b)



q_0	ε	
q_1	$0(10)^n$, $10(10)^n$	$n \geqslant 0$
q_2	$0(10)^n 0w$, $10(10)^n 0w$	$n \geqslant 0$, $w \in \{0,1\}^*$
q_3	$1(01)^n$, $01(01)^n$	$n \geqslant 0$
q_4	$1(01)^n 1w$, $01(01)^n w$	$n \geqslant 0$, $w \in \{0,1\}^*$

q_0	ε
q_1	$(0+10)(10)^*$
q_2	$(0+10)(10)^*0^+$
q_3	$(1+01)(01)^*$
q_4	$(0+10)(10)^*0^+1(0^+1)^*$
q_5	$(1+01)(01)^*1^+$
q_6	$(1+01)(01)^*1^+0(1^+0)^*$
q_7	$(1+01)(01)^*1^+0(1^+0)^*0(0+1)^* +$ $(0+10)(10)^*0^+1)(0^+1)^*1(0+1)^*$

For item (a) we use that, for $n \geqslant 0$, $0(10)^n 0 = (01)^n 00$, $10(10)^n 0 = 1(01)^n 00$, and $1(01)^n 1 = (10)^n 11$, $01(01)^n 1 = 0(10)^n 11$. From this it follows that accepted strings contain the substring 00, if accepted by state q_2, or contain the substring 11, if accepted by state q_4. For item (b), for which verification with the help of path sets was not asked, we have employed so-called regular expressions (discussed in Section 2.3) to describe the path sets. Here we use

$$(1+01)(01)^*1^+ = (\varepsilon+0)1(01)^*1^+ = (\varepsilon+0)(10)^*111^*$$
$$0(1^+0)^*0(0+1)^* = (01^+)^*00(0+1)^*$$

and

$$(0+10)(10)^*0^+ = (\varepsilon+1)0(10)^*0^+ = (\varepsilon+1)(01)^*000^*$$
$$1(0^+1)^*1(1+0)^* = (10^+)^*11(1+0)^*$$

to see that strings accepted by the DFA D'_4 have both a substring 00 and a substring 11.

Exercise 2.1.5. Suppose a language $L \subseteq \Sigma^*$ is accepted by a DFA D. Construct a DFA D^C that accepts the language $L^C = \{ w \in \Sigma^* \mid w \notin L \}$.

Answer to Exercise 2.1.5 Suppose $D = (Q, \Sigma, \delta, q_0, F)$. Define the DFA $D^C = (Q, \Sigma, \delta, q_0, Q \setminus F)$. Thus D and D^C are the same except that a state q is a final state of D^C iff q is not a final state of D.

Now suppose $w = a_1 \cdots a_n \in L$. Since $L = \mathcal{L}(D)$. Let the states $q'_0, \ldots, q'_n$ be such that

$$q'_0 = q_0, \;\; \delta(q'_{i-1}, a_i) = q'_i \text{ for } 1 \leqslant i \leqslant n, \text{ and } q'_n \in F$$

In particular, q'_n is a final state of D. Since $q'_n \notin Q \setminus F$, q'_n is not a final state of D^C. Therefore, $w \notin \mathcal{L}(D^C)$.

Reversely, if $w = a_1 \cdots a_n \notin L$ and $q'_0, \ldots, q'_n$ are such that $q'_0 = q_0$ and $q'_i = \delta(q'_{i-1}, a_i)$ for $1 \leqslant i \leqslant n$, then $q'_n \notin F$. But then $q'_n \in Q \setminus F$, and therefore $w \in \mathcal{L}(D^C)$.

We conclude $w \in \mathcal{L}(D)$ iff $w \notin \mathcal{L}(D^C)$. Put differently, $L = \Sigma^* \setminus \mathcal{L}(D^C)$ or $L^C = \mathcal{L}(D^C)$.

Exercise 2.1.6. Let D_1 and D_2 be two DFAs, say $D_i = (Q_i, \Sigma, \delta_i, q^i_0, F_i)$ for $1 \leqslant i \leqslant 2$.

(a) Give a DFA D with set of states $Q_1 \times Q_2$ and alphabet Σ such that $\mathcal{L}(D) = \mathcal{L}(D_1) \cap \mathcal{L}(D_2)$.

(b) Prove, by induction on the length of a string w, that

$$((q_1, q_2), w) \vdash^n_D ((q'_1, q'_2), w') \iff (q_1, w) \vdash^n_D (q'_1, w') \wedge (q_2, w) \vdash^n_D (q'_2, w')$$

(c) Conclude that indeed $\mathcal{L}(D) = \mathcal{L}(D_1) \cap \mathcal{L}(D_2)$.

Answer to Exercise 2.1.6

(a) Put $D = (Q, \Sigma, \delta, q_0, F)$ where $Q = Q_1 \times Q_2$, $\delta : Q \times \Sigma \to Q$ is such that $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta(q_2, a))$, $q_0 = (q^1_0, q^2_0)$ and $F = F_1 \times F_2$.

(b) Basis, $n = 0$: Clear. It holds that

$$\begin{aligned} &((q_1, q_2), w) \vdash^0_D ((q'_1, q'_2), w') \\ &\iff q'_1 = q_1 \wedge q'_2 = q_2 \wedge w' = w \\ &\iff (q_1, w) \vdash^0_D (q'_1, w') \wedge (q_2, w) \vdash^0_D (q'_2, w') \end{aligned}$$

Induction step, $n > 0$: It holds that

$$\begin{aligned}
& ((q_1, q_2), w) \vdash_D^{n+1} ((q_1', q_2'), w') \\
\iff\ & \exists \bar{q}_1 \in Q_1,\ \exists \bar{q}_2 \in Q_2,\ \exists \bar{w} \in \Sigma^*\colon \\
& \qquad ((q_1, q_2), w) \vdash_D^{n} ((\bar{q}_1, \bar{q}_2), \bar{w}) \vdash_D ((q_1', q_2'), w') \\
\iff\ & \exists \bar{q}_1 \in Q_1,\ \exists \bar{q}_2 \in Q_2,\ \exists a \in \Sigma\colon \\
& \qquad ((q_1, q_2), w) \vdash_D^{n} ((\bar{q}_1, \bar{q}_2), \bar{w}) \wedge \bar{w} = aw' \wedge \delta((\bar{q}_1, \bar{q}_2), a) = (q_1', q_2') \\
\overset{\text{IH}}{\iff}\ & \exists \bar{q}_1 \in Q_1,\ \exists \bar{q}_2 \in Q_2,\ \exists a \in \Sigma\colon \\
& \qquad (q_1, w) \vdash_D^{n} (\bar{q}_1, \bar{w}) \wedge (q_2, w) \vdash_D^{n} (\bar{q}_2, \bar{w}) \wedge \\
& \qquad\qquad \bar{w} = aw' \wedge \delta_1(\bar{q}_1, a) = q_1' \wedge \delta_2(\bar{q}_2, a) = q_2' \\
\iff\ & \exists \bar{q}_1 \in Q_1,\ \exists \bar{q}_2 \in Q_2,\ \exists \bar{w} \in \Sigma^*\colon \\
& \qquad (q_1, w) \vdash_D^{n} (\bar{q}_1, \bar{w}) \wedge (\bar{q}_1, \bar{w}) \vdash_D (q_1', w') \wedge \\
& \qquad\qquad (q_2, w) \vdash_D^{n} (\bar{q}_2, \bar{w}) \wedge (\bar{q}_2, \bar{w}) \vdash_D (q_2', w') \\
\iff\ & (q_1, w) \vdash_D^{n+1} (q_1', w') \wedge (q_2, w) \vdash_D^{n+1} (q_2', w')
\end{aligned}$$

where the induction hypothesis is used at the third equivalence.

(c) We have

$$\begin{aligned}
\mathcal{L}(D) &= \{\, w \in \Sigma^* \mid \exists (q_1, q_2) \in F\colon ((q_0^1, q_0^2), w) \vdash_D^* ((q_1, q_2), \varepsilon) \,\} \\
&\quad \text{(by Definition 2.6)} \\
&= \{\, w \in \Sigma^* \mid \exists (q_1, q_2) \in F\colon ((q_0^1, q_0^2), w) \vdash_D^{|w|} ((q_1, q_2), \varepsilon) \,\} \\
&\quad \text{(by Lemma 2.4)} \\
&= \{\, w \in \Sigma^* \mid \exists q_1 \in F_1\colon (q_0^1, w) \vdash_{D_1}^{|w|} (q_1, \varepsilon) \wedge \exists q_2 \in F_2\colon (q_0^2, w) \vdash_{D_2}^{|w|} (q_2, \varepsilon) \,\} \\
&\quad \text{(by construction of } D\text{)} \\
&= \{\, w \in \Sigma^* \mid \exists q_1 \in F_1\colon (q_0^1, w) \vdash_{D_1}^{|w|} (q_1, \varepsilon) \,\} \cap \\
&\qquad \{\, w \in \Sigma^* \mid \exists q_2 \in F_2\colon (q_0^2, w) \vdash_{D_2}^{|w|} (q_2, \varepsilon) \,\} \\
&\quad \text{(trading conjunction for intersection)} \\
&= \{\, w \in \Sigma^* \mid \exists q_1 \in F_1\colon (q_0^1, w) \vdash_{D_1}^{*} (q_1, \varepsilon) \,\} \cap \\
&\qquad \{\, w \in \Sigma^* \mid \exists q_2 \in F_2\colon (q_0^2, w) \vdash_{D_2}^{*} (q_2, \varepsilon) \,\} \\
&\quad \text{(by Lemma 2.4 again)} \\
&= \mathcal{L}(D_1) \cap \mathcal{L}(D_2) \\
&\quad \text{(by Definition 2.6 twice)}
\end{aligned}$$

2.2 Finite automata

A DFA has a transition function. Thus, each state has exactly one outgoing transition for each symbol. In this section we consider a less strict type of automata. These automata

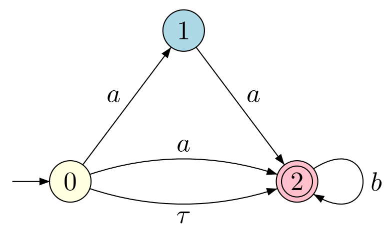


Figure 2.3: Example NFA

may exibit non-determinism as they can have any number of outgoing transitions for a given symbol, including no transitions. Additionally, transitions are allowed that do not consume input. These are silent referred to as silent steps. So, we consider non-deterministic finite automata with silent steps, NFA for short.

Definition 2.10 (Non-deterministic finite automaton with silent steps)**.** A non-deterministic finite automaton with silent steps, or NFA, is a quintuple $N = (Q,\, \Sigma,\, \to_N,\, q_0,\, F\,)$ with Q a finite set of states, Σ a finite alphabet, $\to_N \subseteq Q \times \Sigma_\tau \times Q$ the *transition relation*, $q_0 \in Q$ the initial state, and $F \subseteq Q$ the set of final states.

Instead of a function $\delta : Q \times \Sigma \to Q$ as we have for a DFA, we consider for an NFA a relation $\to_N \;\subseteq\; Q \times \Sigma_\tau \times Q$. This relation may include triples (q, a, q') for states $q, q' \in Q$ and a symbol $a \in \Sigma$, but also triples (q, τ, q') for states $q, q' \in Q$ and the special symbol τ. The symbol τ denotes a so-called silent step. We write Σ_τ for $\Sigma \cup \{\tau\}$. It is assumed that $\tau \notin \Sigma$. We often use α to range over Σ_τ. We write $q \xrightarrow{\alpha}_N q'$ if $(q, \alpha, q') \in \;\to_N$, thus with $\alpha \in \Sigma_\tau$. In case $q \xrightarrow{\tau}_N q'$ we say that there is a silent step or τ-transition in N from q to q'. As we will make explicit below, a silent step does not affect the input of an NFA. We often omit the subcript N when clear from the context.

The concepts of an NFA and of a DFA are very similar, but differ in three aspects:

- in an NFA states can have τ-transitions;
- in an NFA states can have multiple transitions for the same symbol;
- in an NFA states can have no transitions for a symbol.

Figure 2.3 gives a visual representation of a non-deterministic finite automaton, N say. Initial state q_0 has a τ-transition to state q_2, has two transitions on symbol a (one going to q_1, and one going to q_2) and has no transition on symbol b. The transition relation $\to_N$ of N contains the triples

$$(q_0, \tau, q_2) \qquad (q_0, a, q_1) \qquad (q_0, a, q_2) \qquad (q_1, a, q_2) \qquad (q_2, b, q_2)$$

Still we can interpret $\to_N$ as a function$\hat{\delta}_N$, but now $\hat{\delta}_N : Q \times \Sigma_\tau \to \mathcal{P}(Q)$, with $\mathcal{P}(Q)$ the powerset of Q, the collection of all the subsets of Q. Here we have

$$\begin{array}{lll}
\hat{\delta}_N(q_0, a) = \{q_1, q_2\} & \hat{\delta}_N(q_1, a) = \{q_2\} & \hat{\delta}_N(q_2, a) = \emptyset \\
\hat{\delta}_N(q_0, b) = \{q_2\} & \hat{\delta}_N(q_1, b) = \emptyset & \hat{\delta}_N(q_2, b) = \{q_2\} \\
\hat{\delta}_N(q_0, \tau) = \{q_2\} & \hat{\delta}_N(q_1, \tau) = \emptyset & \hat{\delta}_N(q_2, \tau) = \emptyset
\end{array}$$

Note $\hat{\delta}_N(q_0, b) = \{q_2\}$. The idea is to combine the τ-transition $q_0 \xrightarrow{\tau}_N q_2$ with the b-transition $q_2 \xrightarrow{b}_N q_2$. We come back to this idea for the general situation at a later stage.

As for deterministic finite automata we have the notion of a configuration (q, w) for $q \in Q$ and $w \in \Sigma^*$ for an NFA $N = (Q, \Sigma, \rightarrow_N, q_0, F)$. Also here we have the yield or derives relation among configurations. We put

$$(q, w) \vdash_N (q', w') \quad \text{iff} \quad \exists a \in \Sigma\colon q \xrightarrow{a}_N q' \wedge w = aw' \text{ or } q \xrightarrow{\tau}_N q' \wedge w = w'$$

Note, if $q \xrightarrow{\tau}_N q'$ then we have $(q, w) \vdash_N (q', w)$ without change of the input w.

With $\vdash_N^*$ we denote the reflexive-transitive closure of $\vdash_N$. More precisely, we put $(q, w) \vdash_N^0 (q', w')$ if $q = q'$ and $w = w'$, and $(q, w) \vdash_N^{n+1} (q', w')$ if $(q, w) \vdash_N^n (\bar{q}, \bar{w})$, for some $\bar{q} \in Q$, $\bar{w} \in \Sigma^*$, and $(\bar{q}, \bar{w}) \vdash_N (q', w')$. Then $(q, w) \vdash_N^* (q', w')$ if $(q, w) \vdash_N^n (q', w')$ for some $n \geqslant 0$.

Example 2.11. With respect to finite automaton of Figure 2.3 we have, e.g., $(q_0, abb) \vdash_N^* (q_2, \varepsilon)$ and $(q_0, abb) \vdash_N^* (q_1, bb)$. Also $(q_0, abb) \vdash_N^* (q_2, bb)$, $(q_2, bb) \vdash_N^* (q_2, b)$, and $(q_2, b) \vdash_N^* (q_2, \varepsilon)$. In configuration (q_1, bb) the automaton N cannot process the (first) input b; the automaton N is stuck in that configuration. In contrast $(q_0, bb) \vdash_N^* (q_2, \varepsilon)$, since $(q_0, bb) \vdash_N (q_2, bb)$ via the silent step $q_0 \xrightarrow{\tau}_N q_2$, and $(q_2, bb) \vdash_N (q_2, b)$, $(q_2, b) \vdash_N (q_2, \varepsilon)$.

For NFA we have the followng counterpart to Lemma 2.5 for DFA.

Lemma 2.12. For an NFA $N = (Q, \Sigma, \rightarrow_N, q_0, F)$. it holds that

$$(q, w) \vdash_N^* (q', w') \quad \text{iff} \quad (q, wv) \vdash_N^* (q', w'v)$$

for all words w, w', v and states q, q'.

Proof. We prove, by induction on n, $(q, w) \vdash_N^n (q', w')$ iff $(q, wv) \vdash_N^n (q', w'v)$ for $q, q' \in Q$, $w, w', v \in \Sigma^*$.

Basis, $n = 0$: If $(q, w) \vdash_N^0 (q', w')$, then $q = q'$ and $w = w'$. Hence $q = q'$ and $wv = w'v$. So, $(q, wv) \vdash_N^0 (q', w'v)$. If $(q, wv) \vdash_N^0 (q', w'v)$, then $q = q'$ and $wv = w'v$. Hence, $q = q'$ and $w = w'$. So, $(q, w) \vdash_N^0 (q', w')$.

Induction step, $n + 1$: If $(q, w) \vdash_N^{n+1} (q', w')$, then $(q, w) \vdash_N^n (\bar{q}, \bar{w})$ and $(\bar{q}, \bar{w}) \vdash_N (q', w')$ for some $\bar{q} \in Q$, $\bar{w} \in \Sigma^*$. By induction hypothesis, $(q, wv) \vdash_N^n (\bar{q}, \bar{w}v)$. By definition of $\vdash_N$, for suitable $\alpha \in \Sigma_\tau$, $\bar{q} \xrightarrow{\alpha}_N q'$ and $\bar{w} = \alpha w'$ if $\alpha \in \Sigma$, $\bar{w} = w'$ if $\alpha = \tau$. Then $\bar{q} \xrightarrow{\alpha}_N q'$ and $\bar{w}v = \alpha w'v$ if $\alpha \in \Sigma$, $\bar{w}v = w'v$ if $\alpha = \tau$. Thus, $(\bar{q}, \bar{w}v) \vdash_N (q', w'v)$. Since we already observed $(q, wv) \vdash_N^n (\bar{q}, \bar{w}v)$, we conclude $(q, wv) \vdash_N^{n+1} (q', w'v)$.

If $(q, wv) \vdash_N^{n+1} (q', w'v)$, then $(q, wv) \vdash_N^n (\bar{q}, u)$ and $(\bar{q}, u) \vdash_N (q', w'v)$ for some $\bar{q} \in Q$, $u \in \Sigma^*$. By definition of $\vdash_N$ we have, for suitable $\alpha \in \Sigma_\tau$, $\bar{q} \xrightarrow{\alpha}_N q'$ and $u = \alpha w'v$ if $\alpha \in \Sigma$, $u = w'v$ if $\alpha = \tau$. Put $\bar{w} = \alpha w'$ if $\alpha \in \Sigma$, put $\bar{w} = w'$ if $\alpha = \tau$. It follows that $u = \bar{w}v$, $\bar{q} \xrightarrow{\alpha}_N q'$ and $\bar{w} = \alpha w'$ if $\alpha \in \Sigma$, $\bar{w} = w'$ if $\alpha = \tau$. So, $(q, w) \vdash_N^n (\bar{q}, \bar{w})$ and $(\bar{q}, \bar{w}) \vdash_N (q', w')$, which together give $(q, w) \vdash_N^{n+1} (q', w')$. $\square$

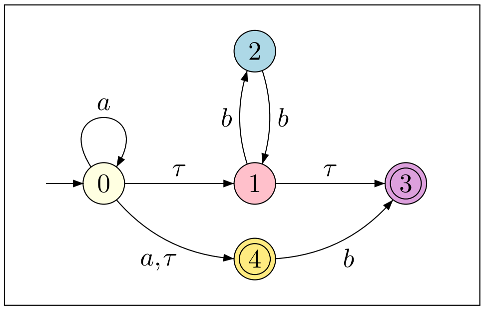


Figure 2.4: NFA of Example 2.14

In the non-deterministic setting of NFAs there is no counterpart of Lemma 2.5, part (a). E.g., for the NFA of Figure 2.3 we have $(q_0, abb) \vdash_N^* (q_1, bb)$ and $(q_0, abb) \vdash_N^* (q_2, bb)$ while $q_1 \neq q_2$.

Definition 2.13 (Language accepted by an NFA)**.** Let $N = (Q, \Sigma, \to_N, q_0, F)$ be a finite automaton. The language $\mathcal{L}(N)$ accepted by N is defined by

$$\mathcal{L}(N) = \{ w \in \Sigma^* \mid \exists q \in F \colon (q_0, w) \vdash_N^* (q, \varepsilon) \}$$

A language accepted by a finite automaton is called a regular language.

Note the similarity of Definition 2.13 for the language of an NFA and the corresponding definition, Definition 2.6, for a DFA. Also note that $\mathcal{L}(N)$ for an NFA N is considered a language over the alphabet Σ, rather than the alphabet Σ_τ. Thus τ-transitions do not contribute (directly) to the word that is accepted.

Example 2.14. Figure 2.4 defines an NFA $N = (Q, \Sigma, \to_N, q_0, F)$ with $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $\Sigma = \{a, b\}$, q_0 as the inital state, set of final states $\{ q_3, q_4 \}$, and transition relation $\to_N$ such that

$$\begin{array}{llll} q_0 \xrightarrow{a}_N q_0 & q_0 \xrightarrow{\tau}_N q_1 & q_0 \xrightarrow{a}_N q_4 & q_0 \xrightarrow{\tau}_N q_4 \\ q_1 \xrightarrow{b}_N q_2 & q_1 \xrightarrow{\tau}_N q_3 & q_2 \xrightarrow{b}_N q_3 & q_4 \xrightarrow{b}_N q_3 \end{array}$$

The accepted language $\mathcal{L}(N)$ of N is given by

$$\{ a^n b,\ a^n (bb)^m \mid n \geqslant 0,\ m \geqslant 0 \}$$

Strings of the format $a^n b$, for $n \geqslant 0$, are accepted via a path of N from the initial state q_0 to the final state q_3 via state q_4. We have

$$(q_0, a^n b) \vdash_N (q_0, a^{n-1} b) \vdash_N \cdots \vdash_N (q_0, b) \vdash_N (q_4, b) \vdash_N (q_3, \varepsilon)$$

Note, the τ-transition $q_0 \xrightarrow{\tau}_N q_4$ involved in $(q_0, b) \vdash_N (q_4, b)$. The a-transition from q_0 to q_4 is superfluous.

Strings of the form $a^n (bb)^m$ are accepted by first processing all a's in state q_0 and next, after reaching state q_1 via the τ-transition, cycling the loop of q_1 to itself via q_2

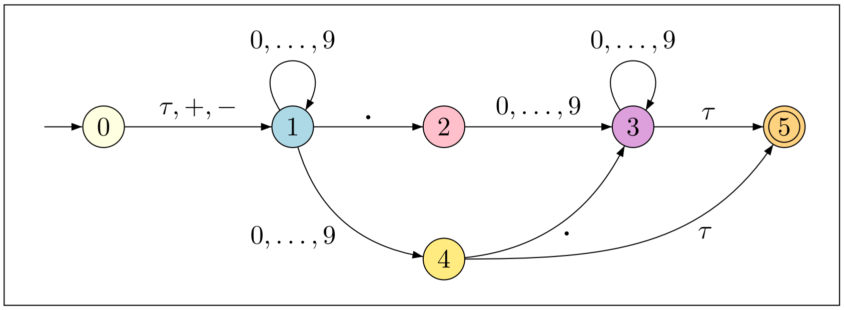


Figure 2.5: An NFA accepting decimal numbers

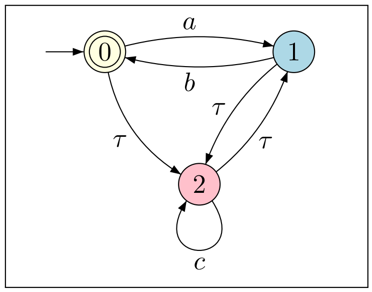


Figure 2.6: NFA of Example 2.16

m times, yielding the substring bb for each time the loop is taken, before moving to q_3 via the second τ-transition. More formally,

$$
\begin{aligned}
&(q_0, a^n(bb)^m) \vdash_N (q_0, a^{n-1}(bb)^m) \vdash_N \cdots \vdash_N (q_0, (bb)^m) \vdash_N \\
&\quad (q_1, (bb)^m) \vdash_N (q_2, b(bb)^{m-1}) \vdash_N (q_1, (bb)^{m-1}) \vdash_N \cdots \vdash_N (q_1, \varepsilon) \vdash_N \\
&\qquad (q_3, \varepsilon)
\end{aligned}
$$

No other computations of N lead to q_3, while at q_3 the automaton is stuck allowing no transitions once q_3 is reached.

Example 2.15. Figure 2.5 provides an NFA that accepts decimal numbers. The three transitions from q_0 to q_1, with labels $+$, $-$ and τ, express that the sign is optional; it can be a $+$-sign, a $-$-sign, or it can be omitted. In state q_1, on reading a digit from input, the automaton can move to state q_4 or stay in state q_1. In state q_4 the τ-transition to state q_5 can always be taken, leading to acceptance. Clearly, this latter transition is redundant, we could have left it out and could have made state q_4 final, instead. For the τ-transition leaving state q_0 it may not be obvious if and how it can be eliminated. However, we shall prove that if a language is accepted by an NFA, then there is a DFA accepting it too.

Example 2.16. Consider the NFA N depicted in Figure 2.6. The language $\mathcal{L}(N)$, the language accepted by N, is given by

$$\begin{aligned}
\mathcal{L}(N) \;=\; \{\, w_1 \cdots w_n \mid n \geqslant 0,\, \forall i, 1 \leqslant i \leqslant n \colon & \\
& \exists n_i \geqslant 0 \exists v_{i,1}, \ldots, v_{i,n_i} \colon \\
& \quad w_i = a v_{i,1} \cdots v_{i,n_i} b \,\wedge\, \forall j, 1 \leqslant j \leqslant n_i \exists m_{i,j} \geqslant 0 \colon v_{i,j} = c^{m_{i,j}} \;\vee \\
& \exists n_i \geqslant 0 \exists v_{i,1}, \ldots, v_{i,n_i} \colon \\
& \quad w_i = v_{i,1} \cdots v_{i,n_i} \,\wedge\, \forall j, 1 \leqslant j \leqslant n_i \exists m_{i,j} \geqslant 0 \colon v_{i,j} = c^{m_{i,j}} \;\; \}
\end{aligned}$$

as can be seen as follows: a string w is accepted if it brings N from the initial state q_0 back to q_0 again, since q_0 is the only final state. Such a string w has the form $w_1 \cdots w_n$ for some $n \geqslant 0$, where each w_i is the labeling of a path from q_0 to itself but not through q_0 in between. So, we need to identify the loops of q_0, going through q_1 and/or q_2, as this determines what strings w_i are allowed.

A loop of q_0 may leave via the a-transition to q_1 and return via the b-transition from q_1 to q_0. In between N may pass through q_1 and q_2 any number of times. This explains the first of the possible string formats $w_i = a v_{i,1} \cdots v_{i,n_i} b$. The paths through q_1 and q_2, leaving from and returning to q_1, are relatively simple. Since the b-transition to q_0 is excluded, N can only move from q_1 to q_2 via a silent step, iterate through the c-loop of q_2, and return via the τ-transition of q_2 back to q_1. This explains the format for the strings $v_{i,j}$, viz. $v_{i,j} = c^{m_{i,j}}$ for some $m_{i,j} \geqslant 0$.

Alternatively, a loop of q_0 may leave via the τ-transition to q_2, but will also now return via the b-transition from q_1 to q_0. This is why the second of the two string formats looks like $w_i = v_{i,1} \cdots v_{i,n_i} b$. Getting to q_1 may take a number of c's, via the loop of q_2, followed by a silent step from q_2 to q_1, i.e. a string c^m in total, where $m \geqslant 0$. From there the analysis above for the loops from q_1 to itself applies. This gives strings of the same format, i.e. c^m, as we have seen. Thus also here $v_{i,j} = c^{m_{i,j}}$.

In Section 2.3 we introduce so-called regular expressions, which allow to describe the language of Example 2.16 more concisely, namely by $\mathcal{L}(N) = \big(ac^*b + c^*b\big)^*$. However, it is clear that in the case of NFA behaviour of the automaton may be more capricious than for DFA, and therefore it may become more difficult to determine the language that is accepted. Below we will present a technique of going from an NFA to an 'equivalent' DFA.

Two NFAs N_1 and N_2 are called language equivalent if they accept the same language, i.e. $\mathcal{L}(N_1) = \mathcal{L}(N_2)$. Likewise for two DFAs. We also call an NFA and a DFA language equivalent if they accept the same language.

Example 2.17. The two finite automata depicted in Figure 2.7 are language equivalent, i.e. they accept the same language. This is the language

$$\{ab, aba\}^* = \{\, w_1 \cdots w_n \mid n \geqslant 0,\, \forall i, 1 \leqslant i \leqslant n \colon w_i = ab \vee q_i = aba \,\}$$

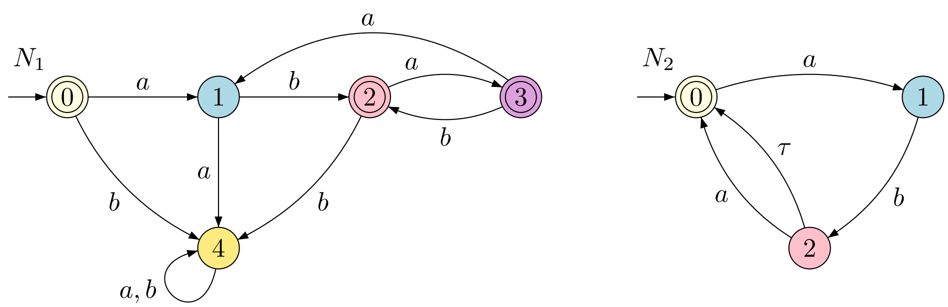


Figure 2.7: Finite automata for Example 2.17

However, the right automaton N_2 allows a τ-transition. The left automaton N_1 can be seen as both as a DFA and as an NFA. The transition relation $\rightarrow_1$ of N_1 is in fact a transition function δ_1. The right automaton N_2 can be seen as an NFA only.

Since a transition function $\delta : Q \times \Sigma \rightarrow Q$ of a DFA can be seen as a transition relation $\rightarrow \subseteq Q \times \Sigma_\tau \times Q$ of an NFA, a DFA can be casted as an NFA with changing the language that is accepted. Therefore, for each DFA there is an language equivalent NFA. Thus, we have the following result.

Theorem 2.18. If a language $L \subseteq \Sigma^*$ is accepted by a DFA, then L is also accepted by an NFA.

Proof. Given a DFA $D = (Q, \Sigma, \delta, q_0, F)$ such that $\mathcal{L}(D) = L$, define the NFA $N = (Q, \Sigma, \rightarrow_N, q_0, F)$ by $q \xrightarrow{a}_N q'$ iff $\delta(q,a) = q'$, for $q, q' \in Q$, $a \in \Sigma$. Then it holds that $(q,w) \vdash_D (q',w')$ iff $(q,w) \vdash_N (q',w')$. Therefore we have, by Definition 2.6 and Definition 2.13, respectively,

$$w \in \mathcal{L}(D) \iff \exists q{\in}F : (q_0, w) \vdash_D^* (q, \varepsilon) \iff \exists q{\in}F : (q_0, w) \vdash_N^* (q, \varepsilon) \iff w \in \mathcal{L}(N)$$

Thus $L = \mathcal{L}(D) = \mathcal{L}(N)$ and L is accepted by the NFA N. □

Note, the NFA N constructed in the proof above does not involve any silent step. The more interesting reverse of Theorem 2.18 holds as well: for each NFA exists an language equivalent DFA.

Theorem 2.19. If a language $L \subseteq \Sigma^*$ is accepted by an NFA, then L is also accepted by a DFA.

Proof. Suppose $L = \mathcal{L}(N)$ for an NFA $N = (Q_N, \Sigma, \rightarrow_N, q_N^0, F_N)$. The so-called ε-closure $E(\bar{q})$ of a state $\bar{q}$ of N is given by

$$E(\bar{q}) = \{ q' \in Q_N \mid (\bar{q}, \varepsilon) \vdash_N^* (q', \varepsilon) \}$$

Thus $q' \in E(\bar{q})$ if there is a sequence of zero, one ore more τ-transitions from $\bar{q}$ to q'. We construct a DFA $D = (Q_D, \Sigma, \delta, Q_D^0, F_D)$ such that $\mathcal{L}(D) = \mathcal{L}(N)$ as follows.

- $Q_D = \mathcal{P}(Q_N)$, i.e. states of D are sets of states of N
- $\delta(Q,a) = \bigcup\{\, E(\bar{q}) \mid q \in Q,\ q \xrightarrow{a}_N \bar{q} \,\}$ for $Q \subseteq Q_N$
- $Q_D^0 = E(q_N^0)$, the ε-closure of the initial state of N
- $F_D = \{\, Q \subseteq Q_N \mid Q \cap F_N \neq \emptyset \,\}$

Put differently, $\delta(Q,a) = \{\, q' \in Q_N \mid \exists q \in Q \colon (q,a) \vdash_N^* (q',\varepsilon) \,\}$, and $Q_D^0 = \{\, q' \in Q_N \mid (q_N^0,\varepsilon) \vdash_N^* (q',\varepsilon) \,\}$. We claim

$$(q,w) \vdash_N^* (q',\varepsilon) \quad \text{iff} \quad \exists Q' \subseteq Q_N \colon (E(q),w) \vdash_D^* (Q',\varepsilon) \text{ and } q' \in Q'$$

Proof of the claim: ($\Rightarrow$) By induction on $|w|$. Basis, $|w| = 0$: Then we have $w = \varepsilon$. Thus $(q,\varepsilon) \vdash_N^* (q',\varepsilon)$, hence $q' \in E(q)$ by definition of $E(q)$. So, for $Q' = E(q)$, we have $(E(q),w) \vdash_D^* (Q',\varepsilon)$ and $q' \in Q'$. Induction step, $|w| > 0$: Then we have $w = va$ for suitable v and a. Thus, for some $\bar{q}, \bar{q}' \in Q_N$,

$$(q,w) \vdash_N^* (\bar{q},a) \vdash_N (\bar{q}',\varepsilon) \vdash_N^* (q',\varepsilon)$$

By Lemma 2.12, $(q,v) \vdash_N (\bar{q},\varepsilon)$. Hence, by induction hypothesis, we can find $\bar{Q} \subseteq Q_N$ such that $(E(q),v) \vdash_D^* (\bar{Q},\varepsilon)$ and $\bar{q} \in \bar{Q}$. By Lemma 2.5 we obtain $(E(q),w) \vdash_D^* (\bar{q},a)$. Since $(\bar{q}',\varepsilon) \vdash_N^* (q',\varepsilon)$, we have $q' \in E(\bar{q}')$. Put $Q' = \bigcup\{\, E(\hat{q}') \mid \hat{q} \in \bar{Q},\ \hat{q} \xrightarrow{a}_N \hat{q}' \,\}$. Then $\bar{Q} \xrightarrow{a}_D Q'$. Note $\bar{q} \in \bar{Q}$ and $q \in Q'$. Combination of all this yields

$$(E(q),w) \vdash_D^* (\bar{Q},a) \vdash_D (Q',\varepsilon) \quad \text{and} \quad q' \in Q'$$

Hence, $(E(q),w) \vdash_D^* (Q',\varepsilon)$ and $q' \in Q'$ as was to be shown.

($\Leftarrow$) By induction on $|w|$. Basis, $|w| = 0$: Then we have $w = \varepsilon$ and $Q' = E(q)$, since D has no τ-transitions. By definition, if $q' \in E(q)$, then $(q,\varepsilon) \vdash_N^* (q',\varepsilon)$. Induction step, $|w| > 0$: Then $w = va$ for suitable v and a. It holds that

$$(E(q),w) \vdash_D^* (\bar{Q},a) \vdash_D (Q',\varepsilon)$$

for some $\bar{Q} \subseteq Q_N$. Thus, again by Lemma 2.5, $(E(q),v) \vdash_D^* (\bar{Q},\varepsilon)$. Since $(\bar{Q},a) \vdash_D (Q',\varepsilon)$ and $q' \in Q'$, we have $\bar{q} \xrightarrow{a}_N \bar{q}'$ for some $\bar{q} \in \bar{Q}$ and $\bar{q}' \in Q_N$ such that $q' \in E(\bar{q}')$. By induction hypothesis, $(q,v) \vdash_N^* (\bar{q},\varepsilon)$ and $(q,w) \vdash_N^* (\bar{q},a)$ by Lemma 2.12. Moverover, $(\bar{q},a) \vdash_N (\bar{q}',\varepsilon)$ and $(\bar{q}',\varepsilon) \vdash_N^* (q',\varepsilon)$. Combining all this yields

$$(q,w) \vdash_N^* (\bar{q},a) \vdash_N (\bar{q}',\varepsilon) \vdash_N^* (q',\varepsilon)$$

Thus $(q,w) \vdash_N^* (q',\varepsilon)$ which proves the claim.

Now, to show that $\mathcal{L}(N) = \mathcal{L}(D)$ we reason as follows:

$$\begin{aligned}
& w \in \mathcal{L}(N) \\
\iff\ & \exists q' \in F_N \colon (q_N^0,w) \vdash_N^* (q',\varepsilon) \\
\iff\ & \exists q' \in F_N\, \exists Q' \subseteq Q_N \colon (E(q_N^0),w) \vdash_D^* (Q',\varepsilon) \wedge q' \in Q' \\
& \hspace{18em} \text{(by the claim)} \\
\iff\ & \exists Q' \in F_D \colon (E(q_N^0),w) \vdash_D^* (Q',\varepsilon) \qquad \text{(if } Q' \in F_D\text{, then } Q' \neq \emptyset) \\
\iff\ & w \in \mathcal{L}(D)
\end{aligned}$$

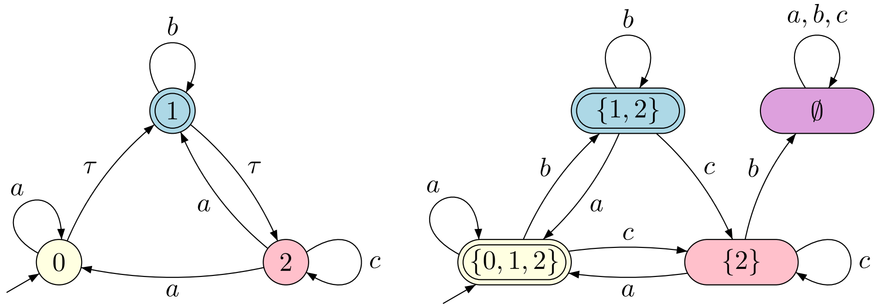


Figure 2.8: An NFA and a language equivalent DFA

This proves the theorem. $\square$

The construction above, in the proof of Theorem 2.19, takes for D the complete power-set $\mathcal{P}(Q_N)$ as set of states. Usually this leads to superfluous states, unreachable from the inital state. To end up with a smaller number of states in D, one can do this more cautiously.

Example 2.20. Consider the automaton N depicted in Figure 2.8 for which we will construct a DFA D accepting the same language. The set of states of D will be built up lazily. Following the construction of Theorem 2.19 we need to include the starting state $E(q_0) = \{q_0, q_1, q_2\}$, the ε-closure of the starting state q_0 of N.

We calculate possible transitions for $\{q_0, q_1, q_2\}$ for all symbols a, b, and c.

$$
\begin{aligned}
q_0 &\xrightarrow{a}_N q_0 \qquad E(q_0) = \{q_0, q_1, q_2\} \\
q_2 &\xrightarrow{a}_N q_0 \qquad E(q_0) = \{q_0, q_1, q_2\} \\
q_2 &\xrightarrow{a}_N q_1 \qquad E(q_1) = \{q_1, q_2\} \\
\text{thus } \{q_0, q_1, q_2\} &\xrightarrow{a}_D \{q_0, q_1, q_2\} \text{ i.e. } \delta(\{q_0, q_1, q_2\}, a) = \{q_0, q_1, q_2\} \\
q_1 &\xrightarrow{b}_N q_1 \qquad E(q_1) = \{q_1, q_2\} \\
\text{thus } \{q_0, q_1, q_2\} &\xrightarrow{b}_D \{q_1, q_2\} \text{ i.e. } \delta(\{q_0, q_1, q_2\}, b) = \{q_1, q_2\} \\
q_2 &\xrightarrow{c}_N q_2 \qquad E(q_2) = \{q_2\} \\
\text{thus } \{q_0, q_1, q_2\} &\xrightarrow{c}_D \{q_2\} \text{ i.e. } \delta(\{q_0, q_1, q_2\}, c) = \{q_2\}
\end{aligned}
$$

Note, apart from the initial state $\{q_0, q_1, q_2\}$ we have encountered two other states, viz.

$\{q_1, q_2\}$ and $\{q_2\}$. We will first calculate the transitions for $\{q_1, q_2\}$.

$$
\begin{array}{ll}
q_2 \xrightarrow{a}_N q_0 & E(q_0) = \{q_0, q_1, q_2\} \\
q_2 \xrightarrow{a}_N q_1 & E(q_1) = \{q_1, q_2\} \\
\multicolumn{2}{l}{\text{thus } \{q_1, q_2\} \xrightarrow{a}_D \{q_0, q_1, q_2\} \text{ i.e. } \delta(\{q_1, q_2\}, a) = \{q_0, q_1, q_2\}} \\
q_1 \xrightarrow{b}_N q_1 & E(q_1) = \{q_1, q_2\} \\
\multicolumn{2}{l}{\text{thus } \{q_1, q_2\} \xrightarrow{b}_D \{q_1, q_2\} \text{ i.e. } \delta(\{q_1, q_2\}, b) = \{q_1, q_2\}} \\
q_2 \xrightarrow{c}_N q_2 & E(q_2) = \{q_2\} \\
\multicolumn{2}{l}{\text{thus } \{q_1, q_2\} \xrightarrow{c}_D \{q_2\} \text{ i.e. } \delta(\{q_1, q_2\}, c) = \{q_2\}}
\end{array}
$$

No new states have been introduced; we continue with calculating the transitions for state $\{q_2\}$.

$$
\begin{array}{ll}
q_2 \xrightarrow{a}_N q_0 & E(q_0) = \{q_0, q_1, q_2\} \\
q_2 \xrightarrow{a}_N q_1 & E(q_1) = \{q_1, q_2\} \\
\multicolumn{2}{l}{\text{thus } \{q_2\} \xrightarrow{a}_D \{q_0, q_1, q_2\} \text{ i.e. } \delta(\{q_2\}, a) = \{q_0, q_1, q_2\}} \\
q_2 \xrightarrow{c}_N q_2 & E(q_2) = \{q_2\} \\
\multicolumn{2}{l}{\text{thus } \{q_2\} \xrightarrow{c}_D \{q_2\} \text{ i.e. } \delta(\{q_2\}, c) = \{q_2\}} \\
\multicolumn{2}{l}{q_2 \text{ has no outgoing } b\text{-transition in } N} \\
\multicolumn{2}{l}{\text{thus } \{q_2\} \xrightarrow{b}_D \emptyset \text{ i.e. } \delta(\{q_2\}, b) = \emptyset}
\end{array}
$$

We choose as set of states Q_D the states that have been introduced up to here. Thus,

$$Q_D \ = \ \{ \, \{q_0, q_1, q_2\}, \, \{q_1, q_2\}, \, \{q_2\}, \, \emptyset \, \}$$

This state is needed to obtain a complete transition function for D. Although the non-determinism of transitions has been resolved now, state $\{q_2\}$ is lacking a transition for b. So, we add $\{q_2\} \xrightarrow{b}_D \emptyset$ together with $\emptyset \xrightarrow{a}_D \emptyset$, $\emptyset \xrightarrow{b}_D \emptyset$, and $\emptyset \xrightarrow{c}_D \emptyset$. Put differently, we have $\delta(\{q_2\}, b) = \emptyset$, $\delta(\emptyset, a) = \emptyset$, $\delta(\emptyset, b) = \emptyset$, and $\delta(\emptyset, c) = \emptyset$. The state $\emptyset$ is called a trap state. Once in, the automaton can't get out from there.

The final states of D are those states in Q_D that, as subsets of Q_N, contain q_1, the single final state of Q_N. These are $\{q_0, q_1, q_2\}$ and $\{q_1, q_2\}$. The resulting DFA D is depicted at the right of Figure 2.8.

Example 2.21. We illustrate the construction of Theorem 2.19 of a DFA D for the NFA N given in Figure 2.4 with accepted language

$$\{ \, a^n b, \, a^n (bb)^m \mid n \geqslant 0, \, m \geqslant 0 \, \}$$

as discussed in Example 2.14.

Say, $N = (Q, \Sigma, \rightarrow_N, q_0, F)$. We put $D = (Q_D, \Sigma, \delta, Q^0, F_D)$. By construction, the initial state Q_0 of D is the ε-closure $E(q_0)$ of the initial state q_0 of N. Thus $Q_0 =$

$E(q_0) = \{q_0, q_1, q_3, q_4\}$, since each state, except q_2 can be reached from q_0 by a sequence of τ-transitions. The alphabet Σ is the same for N and for D.

The theorem proposes to take the complete powerset of N's set of states Q. However, here we do not decide on the set of states yet. Instead, we construct a table encoding the transition function, including states when needed. We start off with state $\{q_0, q_1, q_3, q_4\}$. The a-transitions of N reach q_0 and q_4 from q_0, yielding $\{q_0, q_4\}$. The ε-closure of $\{q_0, q_4\}$ is $\{ q_0, q_1, q_3, q_4 \}$ itself. Thus, $\delta(\{ q_0, q_1, q_3, q_4 \}, a) = \{ q_0, q_1, q_3, q_4 \}$. Starting from $\{ q_0, q_1, q_3, q_4 \}$ the b-transitions of N reach q_2 from q_1 and q_3 from q_4, yielding $\{ q_2, q_3 \}$. The ε-closure of $\{ q_2, q_3 \}$ is $\{ q_2, q_3 \}$, adding no states. Thus $\delta(\{ q_0, q_1, q_3, q_4 \}, b) = \{ q_2, q_3 \}$.

Starting with the inital state $\{ q_0, q_1, q_3, q_4 \}$, we have need of a new state, viz. $\{ q_2, q_3 \}$. We compute the a-transitions and b-transitions for this state. Both q_2 and q_3 have no a-transition in N. Thus, $\delta(\{ q_2, q_3 \}, a) = \emptyset$, which introduces the empty set as a new state of D. Note, the ε-closure of $\emptyset$ is $\emptyset$. The b-transitions for q_2 and q_3 get to q_1 and q_3, respectively, yielding $\{ q_1, q_3 \}$ with ε-closure $\{ q_1, q_3 \}$. Thus $\delta(\{ q_2, q_3 \}, a) = \emptyset$ and $\delta(\{ q_2, q_3 \}, b) = \{ q_1, q_3 \}$.

The empty set will have a transition to itself, for each symbol in the alphabet. Thus, $\delta(\emptyset, a) = \emptyset$, and $\delta(\emptyset, b) = \emptyset$. Therefore, we first consider the newly introduced state $\{ q_1, q_3 \}$ of D. Both q_1 and q_3 have no a-transition in N, thus $\delta(\{ q_1, q_3 \}, a) = \emptyset$. As to b-transitions, q_1 has one, to q_2, q_3 has no. This yields $\{q_2\}$. This set of states is ε closed. Thus, we have a new state $\{q_2\}$ of D, and $\delta(\{ q_1, q_3 \}, b) = \{q_2\}$.

For state q_2 of N there is no a-transition, thus the state $\{q_2\}$ of D has an a-transition to $\emptyset$. The state q_2 has a b-transition in N, though. We have b-transition from q_2 to q_1 which can be extended by the τ-transition to q_3. This gives in D a b-transition to $\{q_1, q_3$. Hence $\delta(\{q_2\}, b) = \{ q_1, q_3 \}$.

Summarizing the above, we have for D the set Q_D of states consisting of the subsets $\{ q_0, q_1, q_3, q_4 \}$, $\{ q_2, q_3 \}$, $\{ q_1, q_3 \}$, $\{q_2\}$ and $\emptyset$, and the transition function $\delta : Q_D \times \{a, b\} \to Q_D$ such that

	a	b
$\{q_0, q_1, q_3, q_4\}$	$\{q_0, q_1, q_3, q_4\}$	$\{q_2, q_3\}$
$\{q_2, q_3\}$	$\emptyset$	$\{q_1, q_3\}$
$\{q_1, q_3\}$	$\emptyset$	$\{q_2, q_3\}$
$\{q_2\}$	$\emptyset$	$\{q_1, q_3\}$
$\emptyset$	$\emptyset$	$\emptyset$

Finally, we have to define the set of final states F_D of D. This are all states in Q_D containing at least one final state of N, i.e. all states containing q_3. Thus, $F_D = \{ \{q_0, q_1, q_3, q_4\}, \{q_2, q_3\}, \{q_1, q_3\} \}$.

The resulting graphical representation of D is given in Figure 2.9. Now, it is more easy to read off the language accepted, viz.

$$\mathcal{L}(D) = \{ a^n,\ a^n b,\ a^n bb(bb)^m \mid n \geqslant 0,\ m \geqslant 0 \}$$

which equals $\mathcal{L}(N) = \{ a^n b,\ a^n (bb)^m \mid n \geqslant 0,\ m \geqslant 0 \}$, with NFA N given by Figure 2.4.

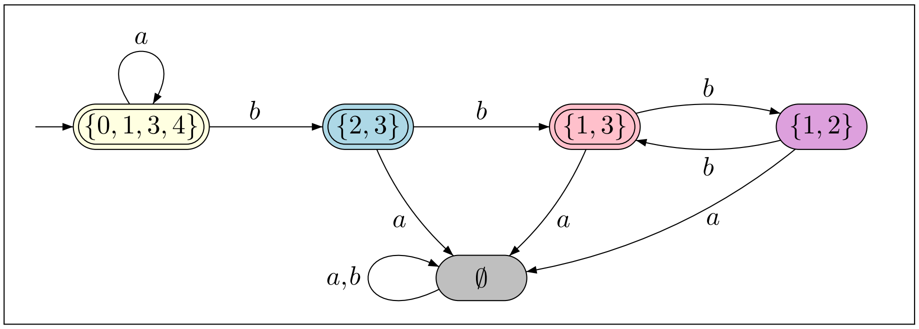


Figure 2.9: A DFA language equivalent to the NFA of Figure 2.4

Exercise 2.2.7. Consider the alphabet $\Sigma = \{a, b, c\}$.

(a) Construct an NFA N_1 that accepts the language

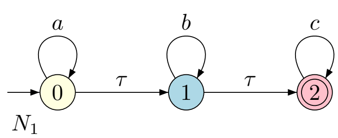
$$L = \{ a^n b^m c^\ell \mid n, m, \ell \geqslant 0 \}$$

and has no more than three states.

(b) Derive a DFA D_1 from the NFA N_1 that accepts L.

Answer to Exercise 2.2.7

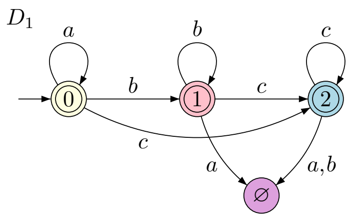
(a)



(b) The ε-closure of q_0 is $\{ q_0, q_1, q_2 \}$, which is the initial state of the DFA. The alphabet is the same as for N_1, hence the alphabet of D_1 is $\{a, b, c\}$. We construct a table of for the transition function of D_1, introducing new states when needed (and writing sequences of numbers to abbreviate sets of states of N_1):

	a	b	c
0, 1, 2	0, 1, 2	1, 2	2
1, 2	$\emptyset$	1, 2	2
2	$\emptyset$	$\emptyset$	2
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

The final states of D_1 are the sets of states containing a final state of N_1, i.e. the sets of states containing q_2. These are $\{q_0, q_1, q_2\}$, $\{q_1, q_2\}$ and $\{q_2\}$. In summary, the DFA D_1 looks as follows.



Exercise 2.2.8. Consider the alphabet $\Sigma = \{a, b, c\}$.

(a) Construct a single NFA N_2 that accepts a string $w \in \Sigma^*$ iff

  (i) w is of the form $ac^n b$ for some $n \geqslant 0$, or

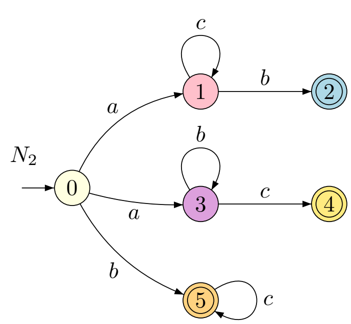
  (ii) w is of the form $ab^m c$ for some $m \geqslant 0$, or

  (iii) w is of the form bc^ℓ for some $\ell \geqslant 0$

(b) Derive a DFA D_2 accepting L from the NFA N_2 of part (a).

Answer to Exercise 2.2.8

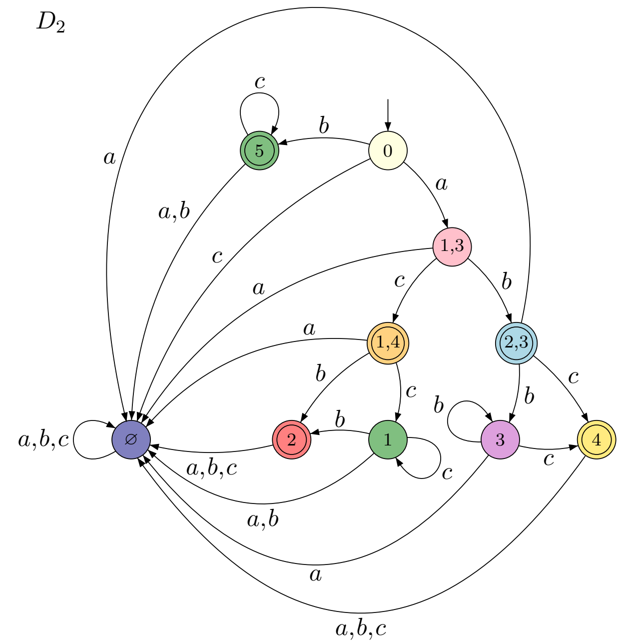
(a)



(b)

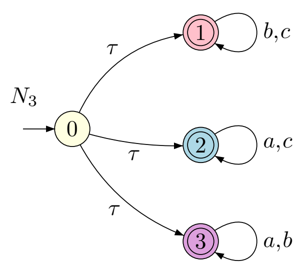
	a	b	c
0	1, 3	5	$\varnothing$
1, 3	$\varnothing$	2, 3	1, 4
5	$\varnothing$	$\varnothing$	5
2, 3	$\varnothing$	3	4
1, 4	$\varnothing$	2	1

	a	b	c
3	$\varnothing$	3	4
4	$\varnothing$	$\varnothing$	$\varnothing$
2	$\varnothing$	$\varnothing$	$\varnothing$
1	$\varnothing$	2	1
$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$



Exercise 2.2.9. Give an automaton over the alphabet $\{a, b, c\}$, with no more than 4 states, that accepts all strings in which at least one symbol of the alphabet does not occur.

Answer to Exercise 2.2.9

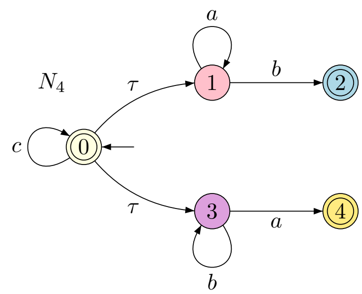


Exercise 2.2.10.

(a) The NFA N_4 below accepts the language

$$\{\, c^n,\ c^n a^m b,\ c^n b^m a,\ c^n \mid n, m \geqslant 0 \,\}$$

Note the empty string $\varepsilon \in \mathcal{L}(N_4)$.

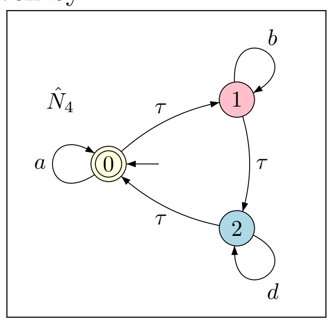


Adapt N_4 to an NFA N_4' that accepts

$$\{ c^n,\ c^n a^m b,\ c^n b^m a \mid n, m \geqslant 0 \}$$

Thus $\mathcal{L}(N_4') = \mathcal{L}(N_4) \setminus \{\varepsilon\}$.

(b) Consider the NFA $\hat{N}_4$ given by

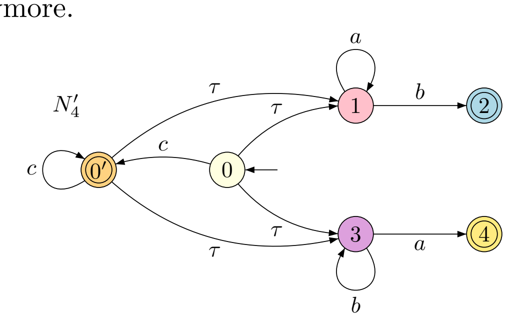


Note $\varepsilon \in \hat{N}_4$. Modify $\hat{N}_4$ into an NFA $\hat{N}_4'$ such that $\mathcal{L}(\hat{N}_4') = \mathcal{L}(\hat{N}_4) \setminus \{\varepsilon\}$.

(c) Suppose the language $L \subseteq \Sigma^*$ is regular and $\varepsilon \in L$. Show that $L \setminus \{\varepsilon\}$ is regular too.

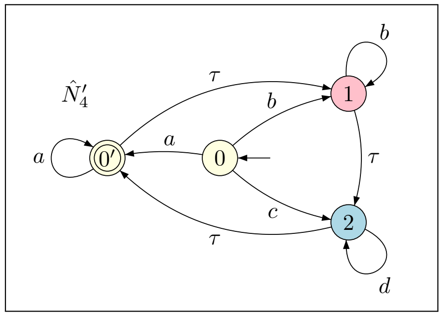
Answer to Exercise 2.2.10

(a) Introduce a new state q_0'. This state has the same transitions as q_0 and is reachable from q_0 by a c-transition, since q_0 has a c-loop. The new state q_0' is accepting, but state q_0 isn't anymore.



(b) We introduce a new final state q_0', while the initial state q_0 is not accepting anymore. Transitions for q_0' are the same as for q_0 in $\hat{N}_4$. Transitions for q_0' are the non-τ

transitions of $E(q_0)$, the ε-closure of q_0. Transitions into q_0 are now redirected to q_0'. In particular, an a-transition from q_0 to q_0' is added because of the a-loop of q_0 in $\hat{N}_4$.



(c) It is easier to do this for a DFA. The language L is regular. So, by definition, there exists an NFA N accepting L. By Theorem 2.19 we can find a language-equivalent DFA D. Thus, $\mathcal{L}(D) = L$. Since $\varepsilon \in L$ the initial state, q_0 say, of D is accepting. Define the DFA $D' = (Q', \Sigma, \delta', q_0, F')$ by (i) $Q' = Q \cup \{q_0'\}$ for a new state q_0'; (ii) δ' is the same as δ, except $\delta'(q,a) = q_0'$ if $\delta(q,a) = q_0$ and $\delta'(q_0',a) = \delta(q_0,a)$ for $q \in Q$, $a \in \Sigma$; (iii) $F' = (F \setminus \{q_0\}) \cup \{q_0'\}$. Then it holds that $\mathcal{L}(D') = \mathcal{L}(D) \setminus \{\varepsilon\} = L \setminus \{\varepsilon\}$. By Theorem 2.18 there exists a language-equivalent NFA N' for D'. It follows that $L \setminus \{\varepsilon\} = \mathcal{L}(D') = \mathcal{L}(N')$ is accepted by an NFA, viz. N', and, hence, that $L \setminus \{\varepsilon\}$ is regular.

Exercise 2.2.11.

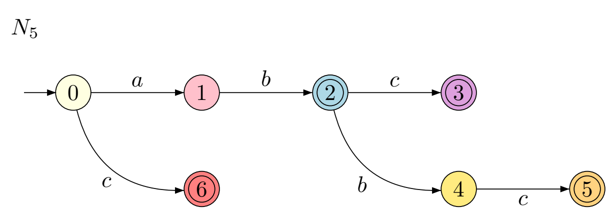
(a) Prove that the language $L = \{abc, abbc, ab, c\}$ is regular.

(b) Construct a DFA accepting L.

(c)6$*$ Prove that every finite language over some alphabet Σ is regular.

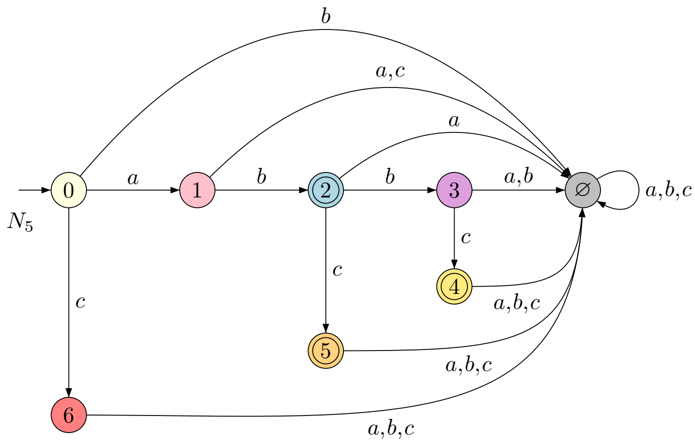
Answer to Exercise 2.2.11

(a) The language L is regular if it is accepted by an NFA. Consider the NFA N_5 below.



Since it accepts L, L is regular.

(b) The following DFA accepts L.



(c)* Suppose $L = \{ w_1, w_2, \ldots, w_n \} \subseteq \Sigma^*$ for some $n \geqslant 0$. Pick, for $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant |w_i|$, symbols $a_{i,j}$ such that $w_i = a_{i,1} a_{i,2} \cdots a_{i,|w_i|}$, for $1 \leqslant i \leqslant n$.

We construct an NFA $N = (Q, \Sigma, \rightarrow_N, q_0, F)$ as follows: Choose pairwise different states q_0 and $q_{i,j}$, for $1 \leqslant i \leqslant n$ and $0 \leqslant j \leqslant |w_i|$. Put

$$\begin{array}{ll} q_0 \xrightarrow{\tau}_N q_{i,0} & \text{for } 1 \leqslant i \leqslant n \\ q_{i,j-1} \xrightarrow{a_{i,j}}_N q_{i,j} & \text{for } 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant |w_i| \end{array}$$

and define $F = \{ q_{i,|w_i|} \mid 1 \leqslant i \leqslant n \}$.

Clearly $w_i \in \mathcal{L}(N)$ for $1 \leqslant i \leqslant n$. Moreover, it holds that $w \in \mathcal{L}(N)$ then $(q_0, w) \vdash_N^* (q_{i,|w_i|}, \varepsilon)$ for some i, $1 \leqslant i \leqslant n$. Then it most hold that $w = a_{i,1} a_{i,2} \cdots a_{i,|w_i|}$, i.e. $w = w_i$. Therefore, $\mathcal{L}(N) = L$ and L is regular.

2.3 Regular expressions

DFAs and NFAs are computational descriptions that are equivalent from a language perspective; both types of automaton accept the same languages, viz. the regular languages. In this section we introduce a syntactic alternative, the regular expressions, that are not based on a computational notion. We will show that for each regular language there is a regular expression that represents is. Reversely, the language associated with a regular expression is always a regular language. It follows that all three notions, (i) language accepted by a DFA, (ii) language accepted by an NFA, and (iii) language belonging to a regular expression amount to the same. After introduction of regular expressions and their associated languages we will prove that for each DFA there exists an equivalent regular expression. Despite its intruiging proof the result is not always convenient in constructing an languaage-equivalent regular expression, e.g. given a DFA. Therefore, we present a procedure to go from a DFA to a regular expression using the notion of a generalized finite automaton. Finally in the section, to complete the claim that regular expressions and regular languages correspond, we show how to obtain an equivalent NFA from a given regular expression.

We start off by introducing the class of regular expressions over a given alphabet. It is built from the constants **0** and **1** as well as constants for every letter from the alphabet under consideration, and it is closed under sum or union, under concatenation and under an operator called Kleene's star or iteration.

Definition 2.22 (Regular expression)**.** Let Σ be an alphabet. The class RE_Σ of regular expressions over Σ is defined as follows.

(i) $\mathbf{1}$ and $\mathbf{0}$ are regular expressions;

(ii) each $a \in \Sigma$ is a regular expression;

(iii) if r_1, r_2 and r are regular expressions, then $(r_1 + r_2)$, $(r_1 \cdot r_2)$ and (r^*) are regular expressions.

When clear from the context, the alphabet Σ may be omitted as subscript of RE_Σ. For reasons to become clear in a minute, the regular expression $(r_1 + r_2)$ called sum, is also called the union of r_1 and r_2, and the regular expression $(r_1 \cdot r_2)$ is called the concatenation of r_1 and r_2. The regular expression (r^*) is called the Kleene-closure or iteration of r.

To reduce the number of parentheses we assume the $*$-construction to bind the most, followed by concatenation $\cdot$, and with sum $+$ having lowest priority. Typically, outermost parentheses will be suppressed too.

Definition 2.23 (Language of a regular expression)**.** Let RE_Σ be the class of regular expressions over the alphabet Σ. The language $\mathcal{L}(r) \subseteq \Sigma$ of a regular expression $r \in RE_\Sigma$ is given by

(i) $\mathcal{L}(\mathbf{1}) = \{\varepsilon\}$ and $\mathcal{L}(\mathbf{0}) = \emptyset$;

(ii) $\mathcal{L}(a) = \{a\}$ for $a \in \Sigma$;

(iii) $\mathcal{L}(r_1 + r_2) = \mathcal{L}(r_1) \cup \mathcal{L}(r_2)$, $\mathcal{L}(r_1 \cdot r_2) = \mathcal{L}(r_1)\mathcal{L}(r_2)$, $\mathcal{L}(r^*) = \mathcal{L}(r)^*$.

In clause (iii), $\mathcal{L}(r_1)\mathcal{L}(r_2)$ denotes the concatenation of $\mathcal{L}(r_1)$ and $\mathcal{L}(r_2)$, while $\mathcal{L}(r)^*$ denotes the Kleene-closure of $\mathcal{L}(r)$. Recall

$$
\begin{aligned}
L_1 L_2 &= \{\, w_1 w_2 \mid w_1 \in L_1,\, w_2 \in L_2 \,\} \\
L^* &= \{\, w_1 \cdots w_k \mid k \geqslant 0,\, w_1, \ldots, w_k \in L \,\}
\end{aligned}
$$

for languages L_1, L_2 and L.

Example 2.24. Consider, with respect to the alphabet $\{a, b\}$, the regular expression $(a + b)^* \cdot a$. We have

$$
\begin{aligned}
&\mathcal{L}((a+b)^* \cdot a) \\
&\quad = \mathcal{L}((a+b)^*)\mathcal{L}(a) \\
&\quad = (\mathcal{L}(a+b))^*\{a\} \\
&\quad = (\mathcal{L}(a) \cup \mathcal{L}(b))^*\{a\} \\
&\quad = (\{a\} \cup \{b\})^*\{a\} \\
&\quad = \{a,b\}^*\{a\} \\
&\quad = \{\, w \in \{a,b\}^* \mid w \text{ ends with } a \,\}
\end{aligned}
$$

Example 2.25. The set of strings over the alphabet $\{a, b, c\}$ that contains at least one a and at most one b can be represented by the regular expression

$$\big(\, c^* \cdot a \cdot (a+c)^* \cdot (b+\mathbf{1}) \cdot (a+c)^* \,\big) + \big(\, c^* \cdot b \cdot c^* \cdot a \cdot (a+c)^* \,\big)$$

The left part of the regular expression covers the situation where at least on symbol a preceeds a *possible* appearance of b. Note, $\mathcal{L}(b+\mathbf{1}) = \{b\} + \{\varepsilon\} = \{b, \varepsilon\}$. The right part represents strings with a single occurrence of b followed by minimally one occurrence of a, interspersed with occurrences of c.

The operators of sum and concatenation are associative with respect to their language interpretation: $\mathcal{L}(r_1 + (r_2 + r_3)) = \mathcal{L}((r_1 + r_2) + r_3))$ and $\mathcal{L}(r_1 \cdot (r_2 \cdot r_3)) = \mathcal{L}((r_1 \cdot r_2) \cdot r_3))$. This allows to write, e.g. $r_1 + r_2 + r_3$, instead of $r_1 + (r_2 + r_3)$ or $(r_1 + r_2) + r_3$, and $r_1 \cdot r_2 \cdot r_3$, since order of bracketing does not matter for the language that is denoted. Often, for notational convenience too, we write $a_1 a_2 \cdots a_n$ for the regular expression $a_1 \cdot a_2 \cdot \cdots \cdot a_n$ using juxtaposition rather than concatenation.

It turns out that for every regular expression r there exists a DFA D_r such that $\mathcal{L}(r) = \mathcal{L}(D_r)$. We will show, see Theorem 2.33 below, that for a regular expression r a language-equivalent NFA $\mathcal{N}_r$ exists. The result then follows by Theorem 2.19. The reverse is valid as well: for every DFA D there exists a regular expression r_D such that $\mathcal{L}(D) = \mathcal{L}(r_D)$. There are various ways to prove this. We first give a conceptually elegant proof based on specific regular expressions $R^k_{i,j}$.

Theorem 2.26. If a language L is accepted by a DFA, then L is the language of a regular expression.

Proof. Suppose $L = \mathcal{L}(D)$ for a DFA $D = (Q, \Sigma, \delta, q_1, F)$ with $Q = \{\, q_1, q_2, \ldots, q_n \,\}$. Note the initial state q_1.

Let the regular expression $R^k_{i,j}$ represent the set of strings w that label the paths from state q_i to state q_j without passing through any state q_ℓ with $\ell > k$ in between.

More precisely, we inductively define $R^k_{i,j}$, for $i, j = 1, \ldots, n$, $k = 0, \ldots, n$, by

$$\begin{array}{lcll} R^0_{i,i} & = & a^1_{i,i} + \cdots + a^{s(i,i)}_{i,i} + \mathbf{1} & \\ R^0_{i,j} & = & a^1_{i,j} + \cdots + a^{s(i,j)}_{i,j} & \text{for } i \neq j \\ R^{k+1}_{i,j} & = & R^k_{i,j} \; + \; R^k_{i,k+1} \cdot (\, R^k_{k+1,k+1}\,)^* \cdot R^k_{k+1,j} & \end{array}$$

where $a^1_{i,j}, \ldots, a^{s(i,j)}_{i,j}$ are all symbols in Σ such that $\delta(q_i, a^1_{i,j}) = q_j, \ \ldots, \ \delta(q_i, a^{s(i,j)}_{i,j}) = q_j$, for $i, j = 1, \ldots, n$.

We verify that the regular expressions $R^k_{i,j}$ are as intended: If $k = 0$ then the path from q_i to q_i has no intermediate states. So, either the path has length 1 if $\delta(q_i, a) = q_i$ yielding the subexpression a with meaning $\{a\}$, or the path has length 0, yielding the subexpression $\mathbf{1}$ with meaning $\{\varepsilon\}$. This explains $R^0_{i,i}$. The explanation of the regular expression $R^0_{i,j}$, for $i \neq j$ is similar. However, in this case there is no empty path from q_i to q_j, since q_i and q_j are different states, so no subexpression $\mathbf{1}$ either.

With respect to the regular expression $R^{k+1}_{i,j}$, a path from q_i to q_j which may pass through intermediate states $q_1, \ldots, q_{k+1}$, may or may not pass through state q_{k+1}. If not, the path is already represented by $R^k_{i,j}$. If the path passes through q_{k+1}, there is an initial part from q_i to q_{k+1} and a final part from q_{k+1} to q_j that both avoid q_{k+1}. In between the path can be split up in subpaths from q_{k+1} to q_{k+1} that do not pass through q_{k+1} as intermediate state; it may pass through intermediate states q_1 to q_k only. The initial part and final part are captured by the regular expressions $R^k_{i,k+1}$ and $R^k_{k+1,j}$, respectively. The subpaths from q_{k+1} to q_{k+1} with intermediate states from q_1 to q_k are included in $R^k_{k+1,k+1}$. Any repetition of such a path is allowed, which explains the Kleene star.

Since the DFA D has n-states, the regular expression $R^n_{i,j}$ represents the labels of all path through D from state q_i to state q_j. We have, for a string w over the alphabet Σ, $w \in \mathcal{L}(D)$ iff there exists a path labeled w from the initial state q_1 of D to a final state $q_f \in F$ iff $w \in \mathcal{L}(R^n_{1,f})$ for a final state q_f. Thus, if $F = \{q_{f_1}, \ldots, q_{f_m}\}$, then $w \in \mathcal{L}(D)$ iff $w \in \mathcal{L}(\, R^n_{1,q_{f_1}} + \cdots R^n_{1,q_{f_m}}\,)$. This proves the theorem, the language L, i.e. $\mathcal{L}(D)$, is the language of the regular expression $R^n_{1,q_{f_1}} + \cdots R^n_{1,q_{f_m}}$. $\square$

Although in principle it is possible to compute $R^n_{1,n}$ from the inductive definition, with n the number of states of the DFA, the method of Theorem 2.26 is not very appealing. As a constructive alternative one can follow a graphical approach for which we need the notion of a generalized finite automaton.

Definition 2.27. A generalized finite automaton $\mathcal{G}$, GFA for short, is a five-tuple $\mathcal{G} = (Q, \Sigma, \delta, q_0, q_e)$ with (i) Q the set of states, (ii) Σ the alphabet, (iii) $\delta : Q \times Q \to RE_\Sigma$ the transition function such that $\delta(q_e, q) = \mathbf{0}$, for all $q \in Q$, (iv) $q_0 \in Q$ the initial state, and (iv) $q_e \in Q$, $q_e \neq q_0$, the (only) final state. The language $\mathcal{L}(\mathcal{G}) \subseteq \Sigma^*$ accepted by GFA $\mathcal{G}$ is given by

$$\begin{aligned} \mathcal{L}(\mathcal{G}) = \{\, w_1 \cdots w_n \in \Sigma^* \mid n \geqslant 1,\, q'_0, \ldots q'_n \in Q \colon & \\ q'_0 = q_0 \wedge q'_n = q_e \wedge \forall i, 1 \leqslant i \leqslant n \colon w_i \in \delta(q_{i-1}, q_i) \,\} & \end{aligned}$$

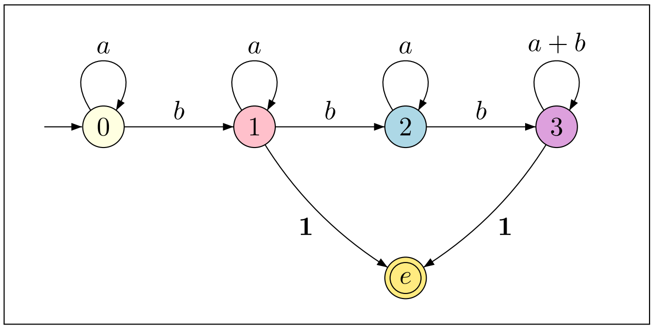


Figure 2.10: GFA of Example 2.28

Note, since $\delta(q_e, q') = \mathbf{0}$ for each $q' \in Q$, we have that $q \neq q_e$ if $w \in \delta(q, q')$ for some string w.

Example 2.28. An example GFA $\mathcal{G}$ is given in Figure 2.10. Here $\mathcal{G} = (Q, \Sigma, \delta, q_0, q_e)$ where set of states $Q = \{q_0, q_1, q_2, q_3, q_e\}$, alphabet $\Sigma = \{a, b\}$, initial state q_0 and final state q_e marked as usual, and with the transition function $\delta \colon Q \times Q \to RE_\Sigma$ given by

	q_0	q_1	q_2	q_3	q_e
q_0	a	b	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$
q_1	$\mathbf{0}$	a	b	$\mathbf{0}$	$\mathbf{1}$
q_2	$\mathbf{0}$	$\mathbf{0}$	a	b	$\mathbf{0}$
q_3	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$a + b$	$\mathbf{1}$
q_e	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$

In the figure we do not draw edges labeled $\mathbf{0}$. Note the edge from q_3 to itself labeled with the regular expression $a + b$, and the edges from q_1 and q_3 to q_e labeled with the regular expression $\mathbf{0}$. The language $\mathcal{L}(\mathcal{G})$ of $\mathcal{G}$ is the set of all strings over $\{a, b\}$ with one, three or more occurrences of the symbol b.

With the notion of a GFA in place, we describe our procedure of constructing a language-equivalent regular expression r_D for a DFA D. As a first step for $D = (Q, \Sigma, \delta, q_0, F)$, we transform D into a language-equivalent GFA G_D as follows: Let q_e be a new state not occurring in Q. Put $Q' = Q \cup \{q_e\}$. We define $\delta' : Q' \times Q' \to RE_\Sigma$ such that

$$\begin{array}{rcll} \delta'(q, q') & = & \delta(q, q') & \text{if } q, q' \neq q_e \\ \delta'(q, q_e) & = & \mathbf{1} & \text{if } q \in F \\ \delta'(q, q_e) & = & \mathbf{0} & \text{if } q \notin F \\ \delta'(q_e, q) & = & \mathbf{0} & \text{for all } q \in Q' \end{array}$$

We call G_D the GFA induced by D.

We argue that the DFA D and the GFA G_D are language-equivalent indeed, i.e. that $\mathcal{L}(D) = \mathcal{L}(G_D)$. For the inclusion $\mathcal{L}(D) \subseteq \mathcal{L}(G_D)$, pick $w \in \mathcal{L}(D)$, say $w = a_1 \cdots a_n$

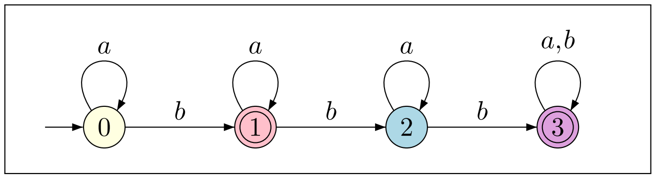


Figure 2.11: DFA of Example 2.29

for $n \geqslant 0$, $a_1, \ldots, a_n \in \Sigma$. Then we can choose $q'_0, \ldots, q'_n \in Q$ such that $q'_0 = q_0$, $\delta(q'_{i-1}, a_i) = q'_i$ for $1 \leqslant i \leqslant n$, and $q'_n \in F$. Put $q'_{n+1} = q_e$. Put $w_i = a_i$ for $1 \leqslant i \leqslant n$ and $w_{n+1} = \varepsilon$. Then we have, for $q'_0, \ldots, q'_{n+1} \in Q'$ that $q'_0 = q_0$, $\delta'(q'_{i-1}, q'_i) = a_i \ni a_i = w_i$ for $1 \leqslant i \leqslant n$ and $\delta'(q'_n, q'_{n+1}) = \mathbf{1} \ni \varepsilon = w_{n+1}$. Thus $w = (a_1 \cdots a_n)\varepsilon = w_1 \cdots w_{n+1} \in \mathcal{L}(G_D)$. For the inclusion $\mathcal{L}(G_D) \subseteq \mathcal{L}(D)$, pick $w \in \mathcal{L}(G_D)$. Then $w = (a_1 \cdots a_n)\varepsilon$, since all edges toward q_e are labeled $\mathbf{1}$. In particular, there are $q'_0, \ldots, q'_n \in Q$ such that $\delta'(q'_{i-1}, q'_i) = a_i$, for $1 \leqslant i \leqslant n$. Moreover, $q'_0 = q_0$ and $q'_n \in F$. By construction it follows that $\delta(q'_{i-1}, a_i) = q_i$, for $1 \leqslant i \leqslant n$, $q'_0 = 0$ and $q'_n \in F$. Thus $w = a_1 \cdots a_n \in \mathcal{L}(D)$, as was to be shown.

Once we have a language-equivalent GFA G_D for a given DFA D, we continue the construction leading to a regular expression r_D for D by successively eliminating intermediate states from G_D, i.e. states different from the inital state q_0 and the final state q_e, one-by-one while updating the labels to ensure language equivalence. When only two states remain, q_0 and q_e, the regular expression r_D such that $\mathcal{L}(D) = \mathcal{L}(r_D)$ can be read off, viz. $r_D = \delta(q_0, q_0)^* \cdot \delta(q_0, q_e)$. Recall, the final state q_e has no outgoing transitions.

Example 2.29. Consider the DFA D as given by Figure 2.11 accepting the language $\{\, w \in \{a,b\}^* \mid \#_a(w) = 1 \vee \#_a(w) \geqslant 3 \,\}$. The GFA G_D, the GFA induced by D, is the GFA of Example 2.28 depicted in Figure 2.10, repeated in the upper-left part of Figure 2.12. The GFA $G_D \backslash p_3$, depicted as upper-right part of Figure 2.12, is obtained from G_D by eliminating state q_3. Since state q_3 of G_D has one incoming transition labeled b, a loop labeled $a + b$, and one outgoing transition labeled $\mathbf{1}$, the path from q_2 to q_e via q_3 is replaced by a transition labeled $b \cdot (a+b)^* \cdot \mathbf{1} = b \cdot (a+b)^*$. Then, removing state q_2 from $G_D \backslash q_3$ yields $(G_D \backslash q_3) \backslash q_2$ at the bottom-left part of Figure 2.12. The path from q_1 via q_2 to q_e is combined with the transition from q_1 to q_e in $G_D \backslash p_3$ giving a single transtion from q_1 to q_e labeled $\mathbf{1} + b \cdot a^* \cdot b \cdot (a+b)^*$. Finally, eliminating state q_1 gives the GFA $((G_D \backslash q_3) \backslash q_2) \backslash q_1$, at the bottom-right of Figure 2.12. The path from q_0 to q_e via q_1 is combined into a single transition from q_0 to q_e labeled by the regular expression $b \cdot a^* \cdot (\mathbf{1} + b \cdot a^* \cdot b \cdot (a+b)^*)$. Because of its simple form, we can read off from $((G_D \backslash q_3) \backslash q_2) \backslash q_1$ the accepted language, viz. $a^* \cdot b \cdot b \cdot a^* \cdot (\mathbf{1} + b \cdot a^* \cdot b \cdot (a+b)^*) = a^* \cdot b \cdot a^* + a^* \cdot b \cdot a^* \cdot b \cdot a^* \cdot b \cdot (a+b)^*$, i.e. the language of strings over $\{a,b\}$ with one, three or more b's. The claim is that the DFA D of Figure 2.29 and the four GFA G_D, $G_D \backslash q_3$, $(G_D \backslash q_3) \backslash q_2$, and $((G_D \backslash q_3) \backslash q_2) \backslash q_1$ of Figure 2.12 are all language equivalent. Hence, the regular expression $a^* \cdot b \cdot a^* + a^* \cdot b \cdot a^* \cdot b \cdot a^* \cdot b \cdot (a+b)^*$ represents the language of the DFA D too.

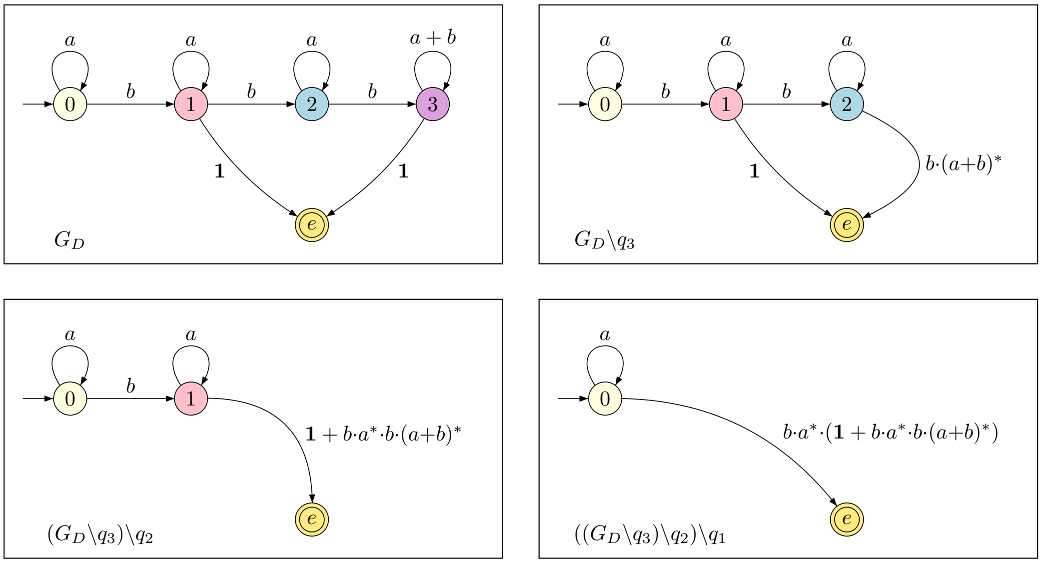


Figure 2.12: GFA sequence G_D, $G_D \backslash q_3$, $(G_D \backslash q_3) \backslash q_2$, $((G_D \backslash q_3) \backslash q_2) \backslash q_1$ of Example 2.28

The approach of successive elimination of states from a GFA, that is language equivalent to a given DFA, to obtain a regular expression for the language of the DFA is justified by the following theorem. Assuming the removal of the state p from a GFA $\mathcal{G}$ gives the GFA $\mathcal{G} \backslash p$, the language of the two automata is the same.

Theorem 2.30. Let $\mathcal{G} = (Q, \Sigma, \delta, q_0, q_e)$ be a GFA. Choose $p \in Q$, $p \neq q_0, q_e$. The GFA $\mathcal{G} \backslash p = (Q', \Sigma, \delta', q_0, q_e)$ with $Q' = Q \backslash \{p\}$ has transition function $\delta' : Q' \times Q' \to RE_\Sigma$ such that

$$\delta'(q, q') = \delta(q, q') \; + \; \delta(q, p) \cdot \big(\delta(p, p)\big)^* \cdot \delta(p, q')$$

if $q, q' \in Q$ and $\delta'(q, q') = \mathbf{0}$ otherwise. Then it holds that $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G} \backslash p)$.

Proof. $(\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\mathcal{G} \backslash p))$ Pick $w = w_1 \cdots w_n \in \mathcal{L}(\mathcal{G})$. Choose $q'_0, \ldots, q'_n \in Q$ such that $q'_0 = q_0$, $q'_n = q_e$, and $w_i \in \delta(q_{i-1}, q_i)$ for $i = 1 \ldots n$. Define strings $v_1, \ldots, v_m \in \Sigma^*$ and states $q''_0, \ldots, q''_m$ as follows: $m = n - \#\{\, k \mid q'_k = p \,\}$, $q''_j = q'_i$ iff $q'_i \neq p$ and $j = i - \#\{\, k \leqslant i \mid q'_k = p \,\}$. Thus, m is the number of states among $q'_0, \ldots, q'_n$ different from p, taking multiplicities into account. Similarly, q''_j is the j-th state in the sequence $q'_0, \ldots, q'_n$ when skipping states q'_k that are equal to p. Define $\ell(j)$, low of j, such that $q''_{j-1} = q'_\ell$ and $h(j)$, high of j, such that $q''_j = q'_h$, for $j = 1 \ldots m$. Note, $q'_k = p$ for $\ell(j) < k < h(j)$, $\ell(1) = 0$ and $h(m) = n$. Now, define $v_j = w_{\ell(j)+1} \cdots w_{h(j)}$. Since $h(j{-}1) = \ell(j)$, for $j = 2 \ldots m$, we have $w = w_1 \cdots w_n = (w_{\ell(1)+1} \cdots w_{h(1)})\,(w_{\ell(2)+1} \cdots w_{h(2)}) \cdots (w_{\ell(m)+1} \cdots w_{h(m)}) = v_1 v_2 \cdots v_m$. Thus for $v_1, \ldots, v_m$ and $q''_0, \ldots, q''_m$ it holds that $w = v_1 \cdots v_m$, $q''_0 = q'_0 = q_0$, $q''_m = q'_n = q_e$, and $v_j \in \delta(q_{\ell(j)}, p) \cdot \big(\delta(p,p)\big)^* \cdot \delta(p, q_{h(j)}) \subseteq \delta'(q''_{j-1}, q''_j)$, for $j = 1 \ldots m$. Thus, $w = v_1 \cdots v_m \in \mathcal{L}(\mathcal{G} \backslash p)$.

$(\mathcal{L}(\mathcal{G} \backslash p) \subseteq \mathcal{L}(\mathcal{G}))$ Suppose $v \in \mathcal{L}(\mathcal{G} \backslash p)$. Pick $q''_0, \ldots, q''_m \in Q \backslash \{p\}$ and $v_1, \ldots, v_m \in \Sigma^*$ such that $q''_0 = q_0$, $q''_m = q_e$, $v = v_1 \cdots v_m$, and $v_j \in \delta'(q''_{j-1}, q''_j)$, for $j = 1 \ldots m$. Since

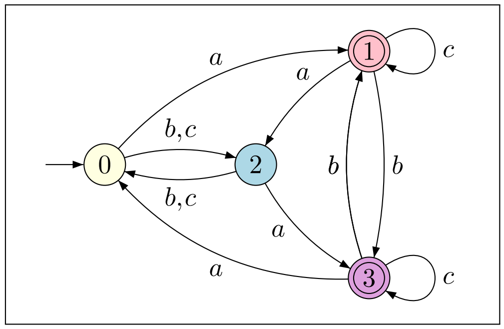


Figure 2.13: DFA of Example 2.32

$\delta'(q''_{j-1}, q''_j) = \delta(q''_{j-1}, q''_j) + \delta'(q''_{j-1}, p) \cdot \big(\delta(p,p)\big)^* \cdot \delta(p, q''_j)$ it follows that we can pick $m(j) \geqslant 0$, and $w_j^0, \ldots, w_j^{m(j)} \in \Sigma^*$ such that $v_j = w_j^0 \cdots w_j^{m(j)}$ where $w_j^0 \in \delta(q''_{j-1}, q''_j)$ if $m(j) = 0$, and $w_j^0 \in \delta(q''_{j-1}, p)$, $w_j^k \in \delta(p,p)$, for $1 \leqslant k < m(j)$, and $w_j^{m(j)} \in \delta(p, q''_j)$ if $m(j) > 0$. It follows, leaving the precise details to the industrious reader, that we can choose $q'_0, \ldots q'_n \in Q$ and $w_1, \ldots, w_n \in \Sigma^*$ such that $q'_0 = q_0$, $q'_n = q_e$, $v = w_1 \ldots w_n$, and $w_i \in \delta(q'_{i-1}, q'_i)$. Thus, $v \in \mathcal{L}(\mathcal{G})$. □

From the theorem we obtain again the result that a language represented by a DFA can also be represented by a regular expression. However, by now we also have a procedure as how to construct the regular expression r_D for the DFA D.

Theorem 2.31. If a language L is accepted by a DFA D, then L is the language of a regular expression r_D.

Proof. We first claim that for an arbitrary GFA $\mathcal{G}$ there exists a regular expression r such that $\mathcal{L}(r) = \mathcal{L}(\mathcal{G})$. We prove the claim by induction on the number of states n of Note, the GFA $\mathcal{G}$ has at least two states, viz. the inital state q_0 and the end state q_e.

Basis, $n = 2$: We have $\mathcal{L}(\mathcal{G}) = \delta(q_0, q_0)^* \cdot \delta(q_0, q_e)$, or more precisely, $\mathcal{L}(G_D) = \mathcal{L}\big(\delta(q_0, q_0)^* \cdot \delta(q_0, q_e)\big)$. So, put $r = \delta(q_0, q_0)^* \cdot \delta(q_0, q_e)$.

Induction step, $n > 2$: Choose a state p different from q_0 and q_e. Consider the GFA $\mathcal{G}\backslash p$ with $n-1$ states. By induction hypothesis, we can choose a regular expression r such that $\mathcal{L}(r) = \mathcal{L}(\mathcal{G}\backslash p)$. By Theorem 2.30, we have $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}\backslash p)$. Therefore, $\mathcal{L}(r) = \mathcal{L}(\mathcal{G})$.

Now, let G_D be the GFA obtained from the DFA D. Then $\mathcal{L}(G_D) = \mathcal{L}(D)$. By the claim we can find a regular expression that we call r_D such that $\mathcal{L}(r_D) = \mathcal{L}(G_D)$. From this we obtain $\mathcal{L}(r_D) = \mathcal{L}(D)$, which proves the theorem. □

In general, there are several ways to reduce the GFA G_D to a two-state GFA. Therefore, there may be several regular expressions that represent the same language.

Example 2.32. As another illustration of the elmination approach to find a regular expression for a DFA, consider the DFA D depicted in Figure 2.13. We start off with the GFA $\mathcal{G}_3$ induced by D, with states q_0 up to q_3 given in the upper-left part of

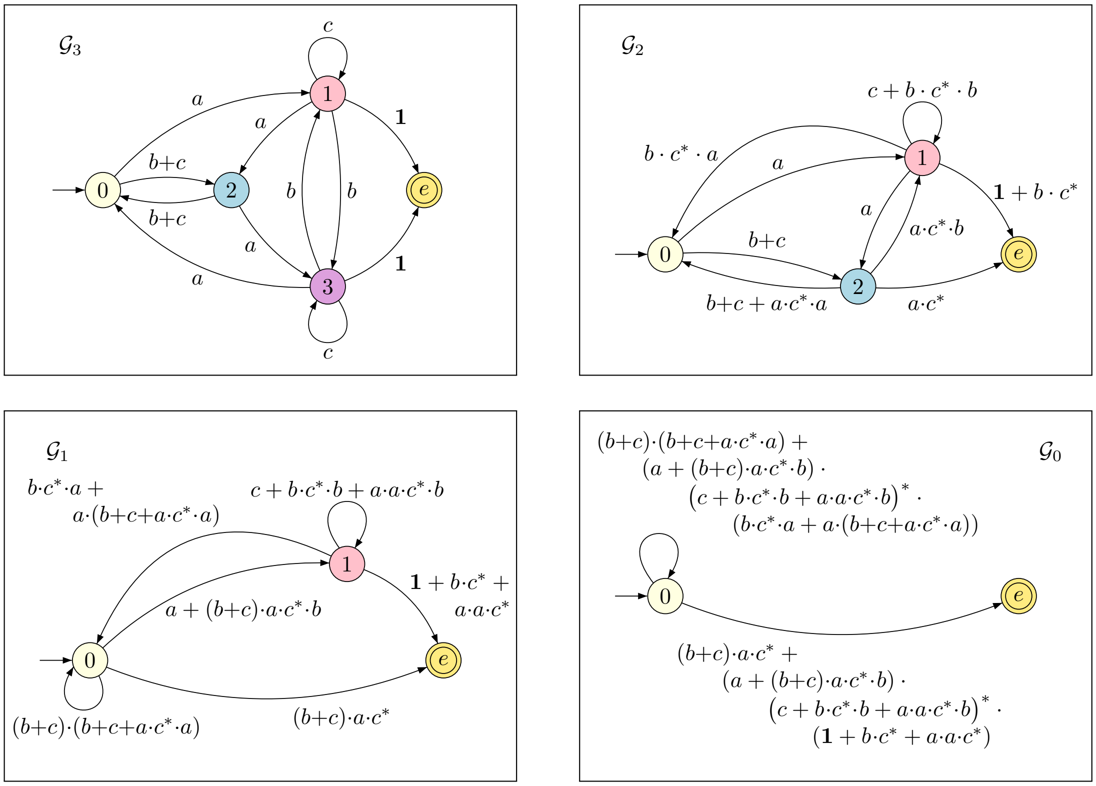


Figure 2.14: GFA sequence of Example 2.32

Figure 2.14. GFA G_D has a single final state, viz. q_e, with incoming edges with label $\mathbf{1}$ from the final states of D. Multiple transitions in D, e.g. from q_0 to q_2, combined into a single edge labeled with a composed regular expression.

We start by eliminating state q_3 from the GFA G_3. In line with the proof of Theorem 2.30, we connect incoming edges of state q_3 with outgoing edges, combined with the self loop of q_3. E.g., since $\delta_3(q_1, q_3) = b$, $\delta_3(q_3, q_3) = c$, and $\delta_3(q_3, q_0) = a$ we create an edge from state q_1 to state q_0 labeled with the regular expression $b \cdot c^* \cdot a$ in the GFA $\mathcal{G}_2$, i.e. $\delta_2(q_1, q_0) = b \cdot c^* \cdot a$. The label of the edge from q_1 to q_e gets strengthened. Since $\delta_3(q_1, q_3) = b$, $\delta_3(q_3, q_3) = c$, and $\delta_3(q_3, q_e) = \mathbf{1}$ and $\delta_3(q_1, q_e) = \mathbf{1}$, we relabel the edge from q_1 to q_e for $\mathcal{G}_2$ with $\mathbf{1} + b \cdot c^* \cdot \mathbf{1} = \mathbf{1} + b \cdot c^*$. Likewise, the edge from q_1 to q_3, combined with the self loop of q_3, and the edge from q_3 to q_1 add the regular expression $b \cdot c^* \cdot b$ to the self loop of q_1, obtaining $c + b \cdot c^* \cdot b$ for $\mathcal{G}_2$. Furthermore, the label of the edge from q_2 to q_0 is updated to $(b + c) + a \cdot c^* \cdot a$, a new edge from q_2 to q_1 is added labeled $a \cdot c^* \cdot b$, as well as another edge from q_2 to q_e labeled $a \cdot c^* \cdot \mathbf{1} = a \cdot c^*$.

The reverse of Theorem 2.31 also holds true: for every regular expression r there exists a DFA accepting the language of r. We will establish this result directly. Instead we show that for every regular expression a corresonding NFA exists. Since for every NFA

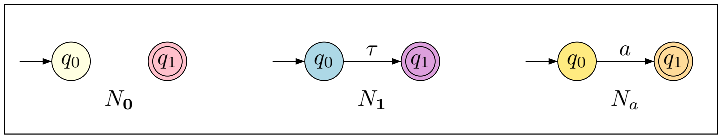


Figure 2.15: NFAs for regular expressions $\mathbf{0}$, $\mathbf{1}$ and a

there exists a language-equivalent DFA, Theorem 2.19, the counterpart of Theorem 2.31 follows.

Theorem 2.33. If a language L equals $\mathcal{L}(r)$ for some regular expression r, then L equals $\mathcal{L}(N)$ for some NFA N.

Proof. Suppose $L \subseteq \Sigma^*$. We prove the theorem by structural induction on the expression r. For each regular expression r we will construct an NFA N_r such that $\mathcal{L}(N_r) = \mathcal{L}(r)$. Moreover, we take care that

(i) N_r has exactly one final state;

(ii) the inital state of N_r has only outgoing transitions (if any);

(iii) the final state of N_r has only incoming transitions.

We need to distinguish three base cases and three successor cases.

Basis, $r = \mathbf{0}$: We have $\mathcal{L}(\mathbf{0}) = \emptyset$. Clearly, $\mathcal{L}(\mathbf{0}) = \mathcal{L}(N_{\mathbf{0}})$ for $N_{\mathbf{0}}$ depicted at the left of Figure 2.15.

Basis, $r = \mathbf{1}$: We have $\mathcal{L}(\mathbf{1}) = \{\varepsilon\}$. Clearly, $\mathcal{L}(\mathbf{1}) = \mathcal{L}(N_{\mathbf{1}})$ for $N_{\mathbf{1}}$ depicted in the center of Figure 2.15.

Basis, $r = a$ for $a \in \Sigma$: We have $\mathcal{L}(a) = \{a\}$. Clearly, $\mathcal{L}(a) = \mathcal{L}(N_a)$ for N_a depicted at the right of Figure 2.15.

Induction step, $r = r_1 + r_2$: Suppose NFA N_i accepts the language $\mathcal{L}(r_i)$, say $N_i = (Q_i,\, \Sigma,\, \delta_i,\, q_0^i,\, \{q_f^i\})$, for $i = 1, 2$. Moverover, we assume $Q_1 \cap Q_2 = \emptyset$. Let q_0 and q_f be two fresh states. Then we put $N_r = (Q,\, \Sigma,\, \delta,\, q_0, \{q_f\})$ where $Q = Q_1 \cup Q_2 \cup \{q_0, q_f\}$ and δ extends δ_1 and δ_2 with $q_0 \xrightarrow{\tau} q_0^1$, $q_0 \xrightarrow{\tau} q_0^2$ and $q_f^1 \xrightarrow{\tau} q_f$, $q_f^2 \xrightarrow{\tau} q_f$. Clearly, by construction, $\mathcal{L}(N_r) = \mathcal{L}(N_1) \cup \mathcal{L}(N_2) = \mathcal{L}(r_1) \cup \mathcal{L}(r_2) = \mathcal{L}(r_1 + r_2)$. See the left-upper part of Figure 2.16.

Induction step, $r = r_1 \cdot r_2$: Suppose NFA N_i accepts the language $\mathcal{L}(r_i)$, say $N_i = (Q_i,\, \Sigma,\, \delta_i,\, q_0^i,\, \{q_f^i\})$, for $i = 1, 2$. Again, we assume $Q_1 \cap Q_2 = \emptyset$. We put $N_r = (Q_1 \cup Q_2,\, \Sigma,\, \delta,\, q_0^1, \{q_f^2\})$ where δ extends δ_1 and δ_2 with $q_f^1 \xrightarrow{\varepsilon} q_0^2$. Clearly, by construction, $\mathcal{L}(N_r) = \mathcal{L}(N_1) \cdot \mathcal{L}(N_2) = \mathcal{L}(r_1) \cdot \mathcal{L}(r_2) = \mathcal{L}(r_1 \cdot r_2)$. See the right-upper part of Figure 2.16.

Induction step, $r = r_0^*$: Suppose NFA N_0 accepts the language $\mathcal{L}(r_0)$, say $N_0 = (Q_0,\, \Sigma,\, \delta_0,\, q_0^0,\, \{q_f^0\})$. Let q_0 and q_f be two fresh states. Then we put $N_r = (Q_0 \cup \{q_0, q_f\},\, \Sigma,\, \delta,\, q_0, \{q_f\})$ where δ extends δ_0 with $q_0 \xrightarrow{\tau} q_0^0$, $q_0 \xrightarrow{\tau} q_f$, and $q_f^0 \xrightarrow{\tau} q_0^0$. Clearly, by construction, $\mathcal{L}(N_r) = \mathcal{L}(N_0)^* = \mathcal{L}(r_0)^* = \mathcal{L}(r_0^*)$. See the lower part of Figure 2.16. $\square$

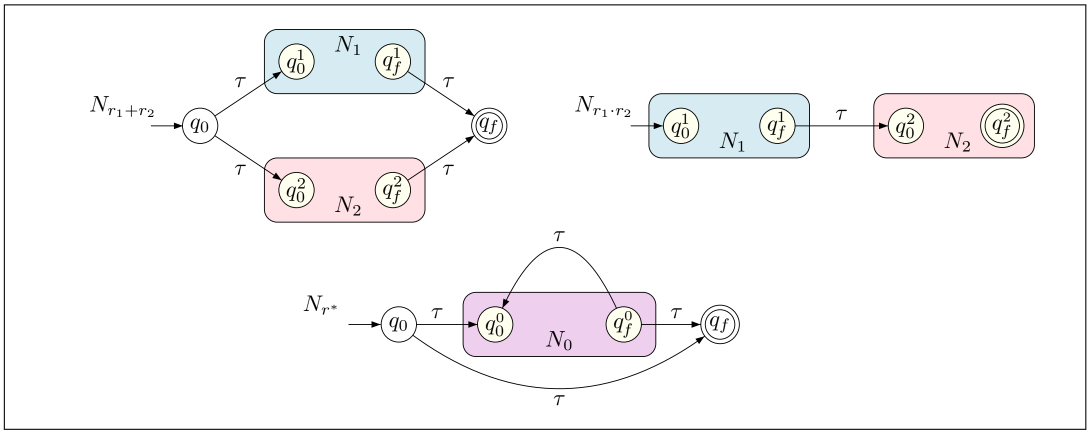


Figure 2.16: NFA for regular expressions $r_1 + r_2$, $r_1 \cdot r_2$ and r^*

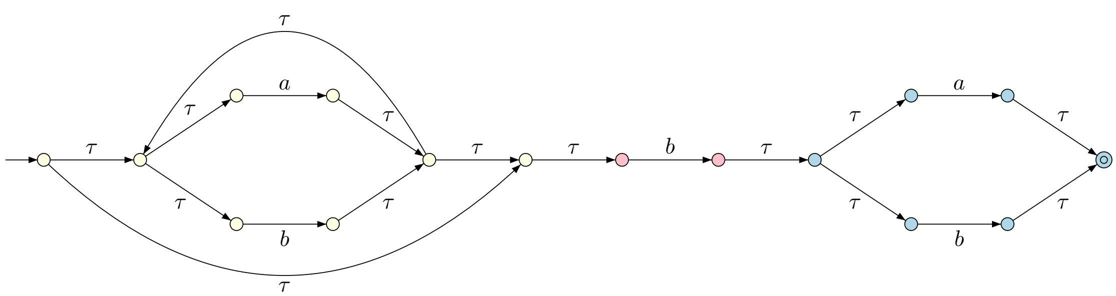


Figure 2.17: NFA of Example 2.34

Example 2.34. Figure 2.17 depicts the NFA obtained by the construction described in the proof of Theorem 2.33 for the regular expression $(a + b)^* \cdot b \cdot (a + b)$. The resulting NFA consists of three sequential components, viz. for the iteration $(a+b)^*$ on the left, for the basic regular expression b in the middle, and for the alternative composition $(a + b)$ on the right.

Exercises for Section 2.3

Exercise 2.3.12. Give for each regular expression r below two strings in $\mathcal{L}(r)$ and two strings not in $\mathcal{L}(r)$, if possible.

(i)	$a^* \cdot b$	(v)	$a^* + b^*$
(ii)	$a \cdot b^* \cdot a$	(vi)	$(a + b) \cdot (a + b)^*$
(iii)	$(\mathbf{1} + a) \cdot b^* \cdot (\mathbf{1} + a)$	(vii)	$(a + b) \cdot (a^* + b^*)$
(iv)	$(ab)^* \cdot (ba)^*$	(viii)	$(a^* + b^*) \cdot (a^* + b^*)$

Answer to Exercise 2.3.12 Possible answers include: (i) b and ab in, ba and aba out; (ii) aa and aba in, ε and ab out; (iii) ε and ab in, bab and aab out; (iv) ε and $abba$ in, a

and aa out; (v) ε and a in, ab and ba out; (vi) a and b in, ε the ony string out; (vii) a and b in, ε and aab out; (viii) ε and a in, aba and bab out.

Exercise 2.3.13. Provide a regular expression for each of the following languages.

(i) $\{\, w \in \{a,b\}^* \mid w \text{ starts with } a \text{ and ends in } b \,\}$

(ii) $\{\, w \in \{a,b,c\}^* \mid w \text{ contains at most two } a\text{'s and at least one } b \,\}$

(iii) $\{\, w \in \{a,b\}^* \mid |w| \leqslant 3 \,\}$

Answer to Exercise 2.3.13

(i) $a \cdot (a+b)^* \cdot b$;

(ii) $c^* \cdot b \cdot (b+c)^* \cdot a \cdot (b+c)^* \cdot a \cdot (b+c)^* + c^* \cdot a \cdot c^* \cdot b \cdot (b+c)^* \cdot a \cdot (b+c)^* + c^* \cdot a \cdot c^* \cdot a \cdot c^* \cdot b \cdot (b+c)^*$;

(iii) $\mathbf{1} + (a+b) + (a+b) \cdot (a+b) + (a+b) \cdot (a+b) \cdot (a+b)$.

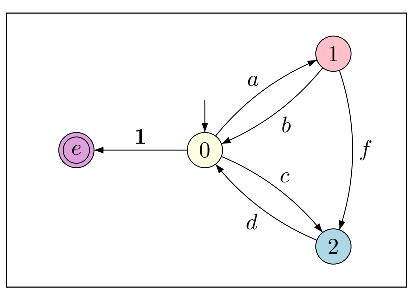
Exercise 2.3.14. Let the GFA $\mathcal{G}$ have two states, q_0 and q_e. Show that it holds that $\mathcal{L}(\mathcal{G}) = \delta(q_0,q_0)^* \cdot \delta(q_0,q_e)$.

Answer to Exercise 2.3.14 $(\mathcal{L}(\mathcal{G}) \subseteq \delta(q_0,q_0)^* \cdot \delta(q_0,q_e))$ Let $w \in \mathcal{L}(\mathcal{G})$. Pick $q'_0, \ldots, q'_n \in Q$ and $w_1, \ldots, w_n \in \Sigma^*$ such that $w = w_1 \cdots w_n$, $q'_0 = q_0$, $w_i \in \delta(q'_{i-1}, q'_i)$ for $1 \leqslant i \leqslant n$, and $q'_n = q_e$. Since $\delta(q_e,q_0) = \delta(q_e,q_0) = \mathbf{0}$ and $\mathcal{L}(\mathbf{0}) = \emptyset$, we have $q'_0, \ldots, q'_{n-1} = q_0$. Thus $w_1, \ldots, w_{n-1} \in \delta(q_0,q_0)$ and $w_n \in \delta(q_0,q_e)$. Hence $w = w_1 \cdots w_n \in \delta(q_0,q_0)^* \cdot \delta(q_0,q_e)$.

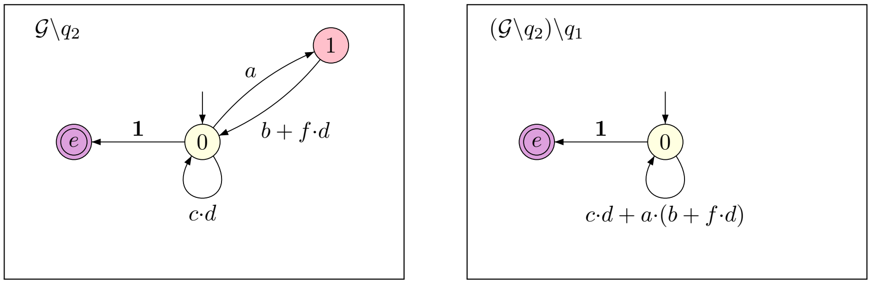
$(\delta(q_0,q_0)^* \cdot \delta(q_0,q_e) \subseteq \mathcal{L}(\mathcal{G}))$ Let $w \in \delta(q_0,q_0)^* \cdot \delta(q_0,q_e)$. Choose $w_1, \ldots, w_n \in \delta(q_0,q_0)$, $w' \in \delta(q_0,q_e)$ such that $w = w_1 \cdots w_n w'$. Put $q'_i = q_0$ for $0 \leqslant i \leqslant n$ and $q'_{n+1} = q_e$. Since $w_i \in \delta(q_0,q_0) = \delta(q_{i-1},q_i)$ for $1 \leqslant i \leqslant n$, and $w' \in \delta(q_0,q_e) = \delta(q'_n, q'_{n+1})$, it follows that $w = w_1 \cdots w_n w' \in \mathcal{L}(\mathcal{G})$.

Exercise 2.3.15. Construct a four-state GFA $\mathcal{G}$, with intermediate states p and q, such that the regular expression belonging to $(\mathcal{G}\backslash p)\backslash q$ is different from the regular expression belonging to $(\mathcal{G}\backslash q)\backslash p$.

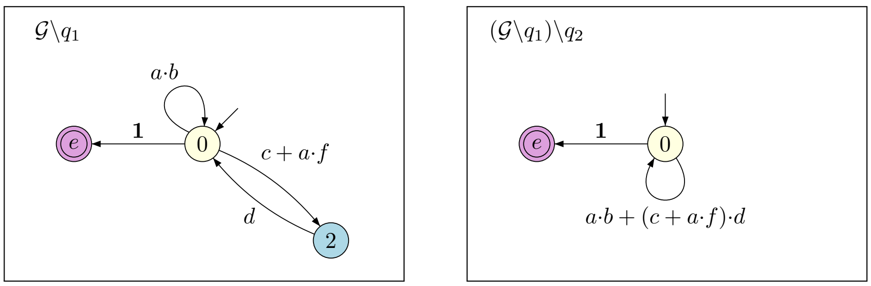
Answer to Exercise 2.3.15 A possible solution is the GFA $\mathcal{G}$:



Elimination of state q_2 first, followed by elimination of state q_1 yields the GFA sequence

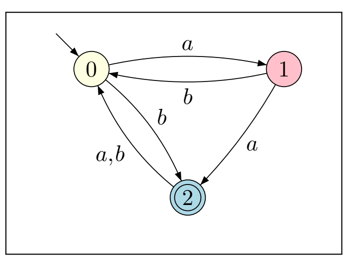


yielding the regular expression $r_{2,1} = \big(c{\cdot}d + a{\cdot}(b + f{\cdot}d)\big)^*$. However, elimination of state q_1 first, followed by elimination of state q_2 yields the following two GFA:

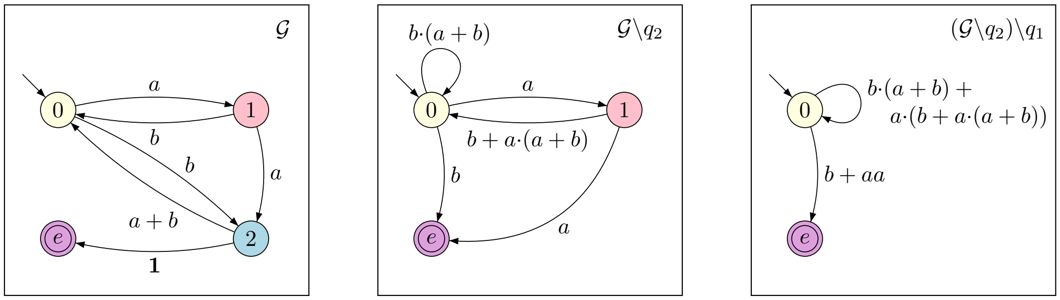


yielding the regular expression $r_{1,2} = \big(a{\cdot}b + (c + a{\cdot}f){\cdot}d\big)^*$. In line with the proof of Theorem 2.31, both regular expressions represent the same language. It holds that $\mathcal{L}(r_{1,2}) = \mathcal{L}(r_{2,1}) = \{\, ab,\ afd,\ cd \,\}^*$.

Exercise 2.3.16. Give a language-equivalent GFA $\mathcal{G}$ for the DFA $\mathcal{D}$ below, successively eliminate the intermediate states of $\mathcal{G}$, and derive a regular expression $r_{\mathcal{D}}$ that represents the language of $\mathcal{D}$.

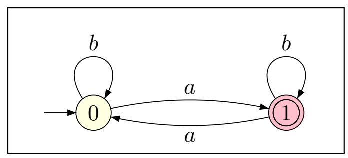


Answer to Exercise 2.3.16 We have the following GFA sequence of $\mathcal{G}$, $\mathcal{G} \backslash q_2$ and $(\mathcal{G} \backslash q_2) \backslash q_1$.



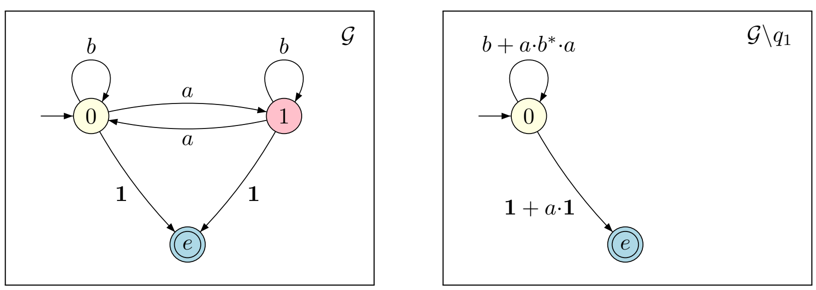
From $(\mathcal{G}\backslash q_2)\backslash q_1$ we obtain $r_{\mathcal{D}} = \big(b \cdot (a+b) + a \cdot (b + a \cdot (a+b))\big)^* \cdot (b + aa)$.

Exercise 2.3.17. (a) Give a language-equivalent GFA $\mathcal{G}$ for the DFA $\mathcal{D}$ below, successively eliminate the intermediate states of $\mathcal{G}$, and derive a regular expression $r_{\mathcal{D}}$ that represents the language of $\mathcal{D}$.



(b) Give a simple regular expression for $\mathcal{L}(\mathcal{D})$.

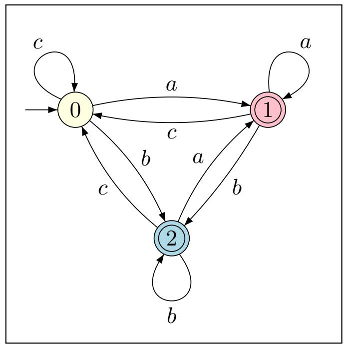
Answer to Exercise 2.3.17 (a) The GFA $\mathcal{G}$ and $\mathcal{G}\backslash q_1$ look as follows



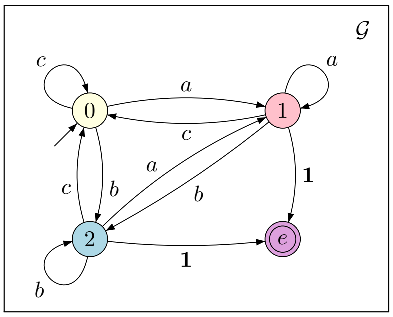
From $\mathcal{G}\backslash q_1$ we obtain $r_{\mathcal{D}} = \big(b + a \cdot b^* \cdot a\big)^* \cdot (\mathbf{1} + a \cdot \mathbf{1})$.

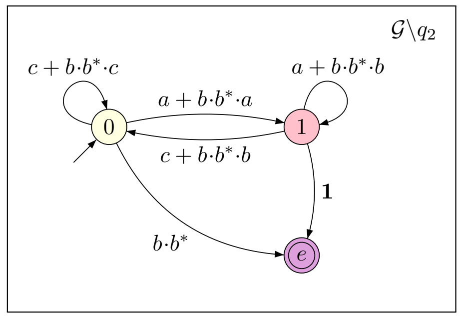
(b) Since $\mathcal{D}$ accepts all strings over $\{a,b\}$, hence $\mathcal{L}(\mathcal{D}) = \{a,b\}^*$, the regular expression $(a+b)^*$ represents $\mathcal{D}$, i.e. $\mathcal{L}((a+b)^*) = \mathcal{L}(\mathcal{D})$.

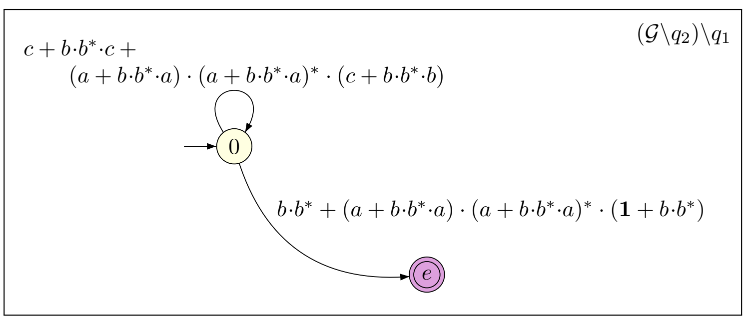
Exercise 2.3.18. Give a language-equivalent GFA $\mathcal{G}$ for the DFA $\mathcal{D}$ below, successively eliminate the intermediate states of $\mathcal{G}$, and derive a regular expression $r_{\mathcal{D}}$ that represents the language of $\mathcal{D}$.



Answer to Exercise 2.3.18 We have the following GFA sequence of $\mathcal{G}$, $\mathcal{G}\backslash q_2$ and $(\mathcal{G}\backslash q_2)\backslash q_1$.







From $(\mathcal{G}\backslash q_2)\backslash q_1$ we obtain $r_{\mathcal{D}} = \big(c + b{\cdot}b^*{\cdot}c + (a + b{\cdot}b^*{\cdot}a) \cdot (a + b{\cdot}b^*{\cdot}a)^* \cdot (c + b{\cdot}b^*{\cdot}b)\big)^* \cdot \big(b{\cdot}b^* + (a + b{\cdot}b^*{\cdot}a) \cdot (a + b{\cdot}b^*{\cdot}a)^* \cdot (\mathbf{1} + b{\cdot}b^*)\big)$.

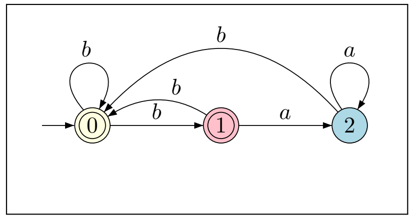
Exercise 2.3.19. Guess a regular expression for each of the following languages. Next provide a DFA for each language and construct a regular expression via elimination of states.

(a) $\{\, w \in \{a,b\}^* \mid$ in w, each maximal substring of a's of length 2 or more is followed by a symbol $b \,\}$

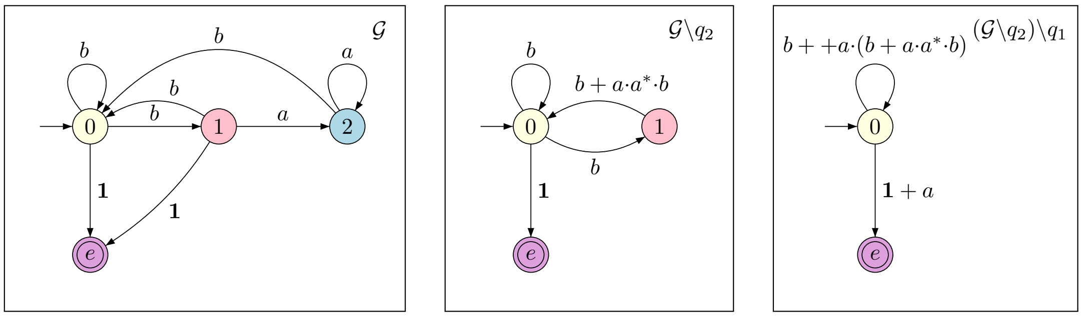
(b) $\{\, w \in \{a,b\}^* \mid w$ has no substring $bab \,\}$

(c) $\{ w \in \{a,b\}^* \mid \#_a(w) = \#_b(w) \wedge v \preccurlyeq w \implies -2 \leqslant \#_a(v) - \#_b(v) \leqslant 2 \}$

Answer to Exercise 2.3.19 (a) The DFA $\mathcal{D}$, given by

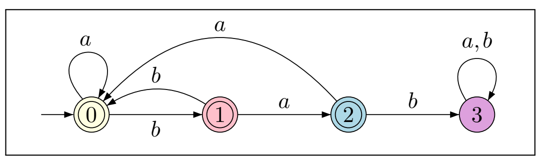


yields the GFA sequence

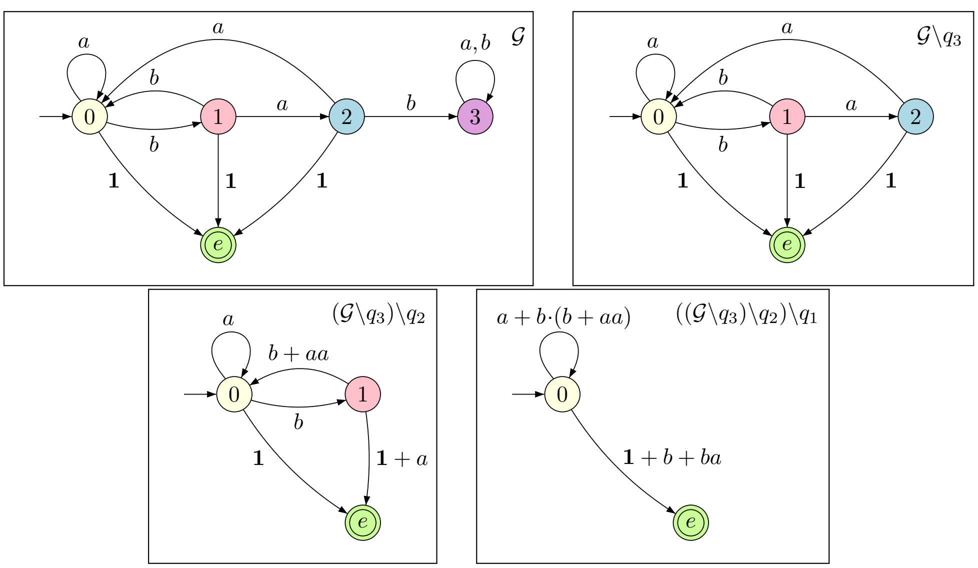


and the regular expression $\big(b + a \cdot (b + a \cdot a^* \cdot b)\big)^*(\mathbf{1} + a)$.

(b) The DFA $\mathcal{D}$, given by

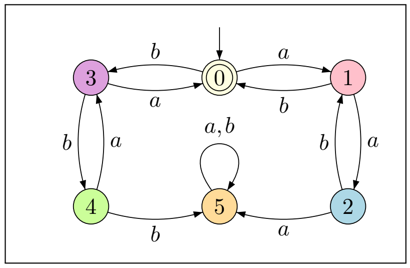


yields the GFA sequence

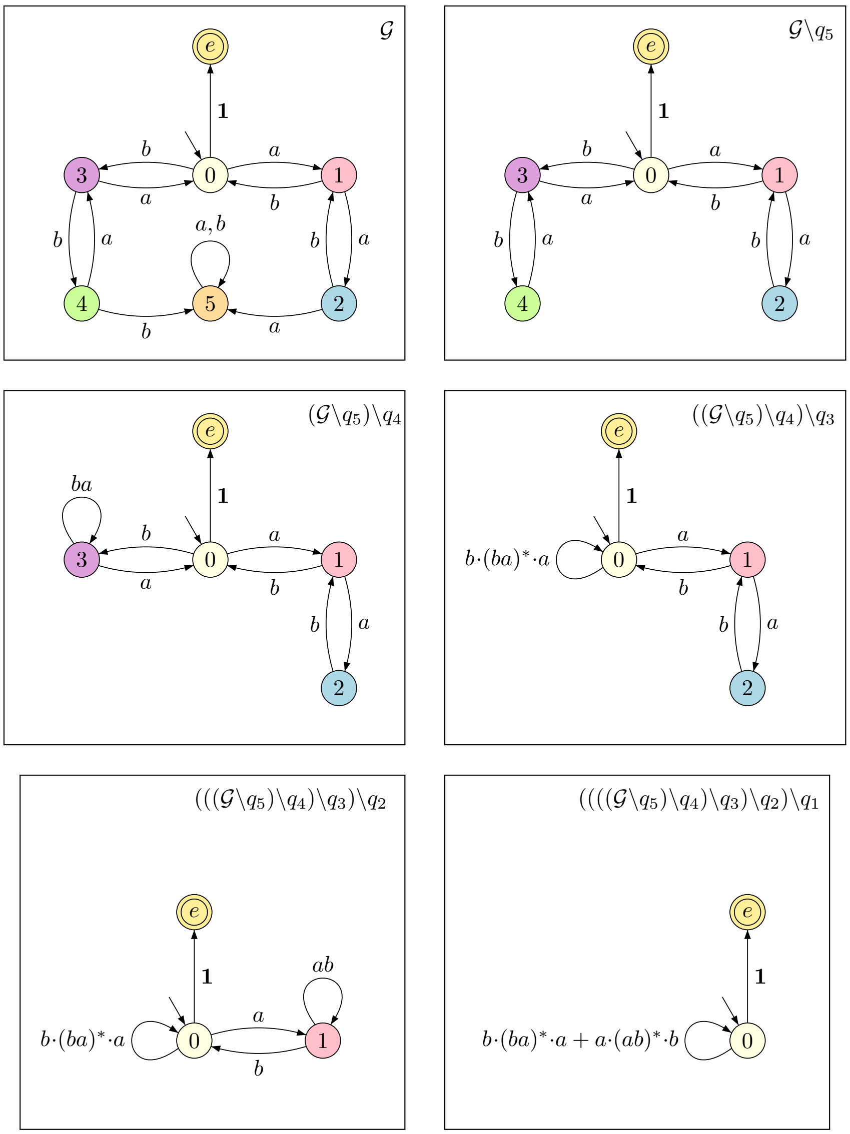


and the regular expression $\big(a + b \cdot (b + aa)\big)^* \cdot (\mathbf{1} + b + ba)$.

(c) The DFA $\mathcal{D}$, given by



yields the GFA sequence



and the regular expression $\big((b \cdot (ba)^* \cdot a) + (a \cdot (ab)^* \cdot b)\big)^*$.

Exercise 2.3.20. Construct for each of the following regular expressions a language-equivalent NFA.

(a) $a \cdot (b + c) \cdot d^*$

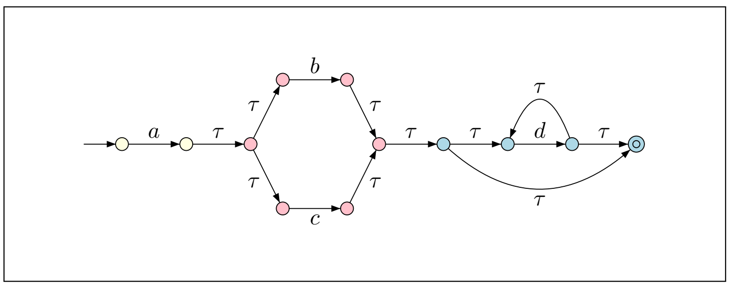
(b) $(a + b) \cdot c^* \cdot d$

(c) $(a + b)^* \cdot c \cdot (d + e)^*$

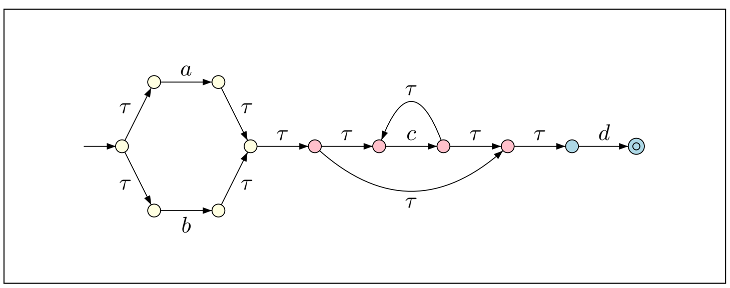
(d) $\big((a + b)^* + c\big)^* + d$

Answer to Exercise 2.3.20

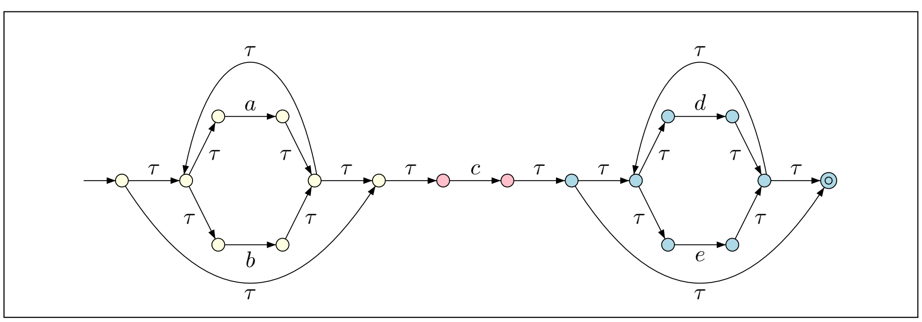
(a)



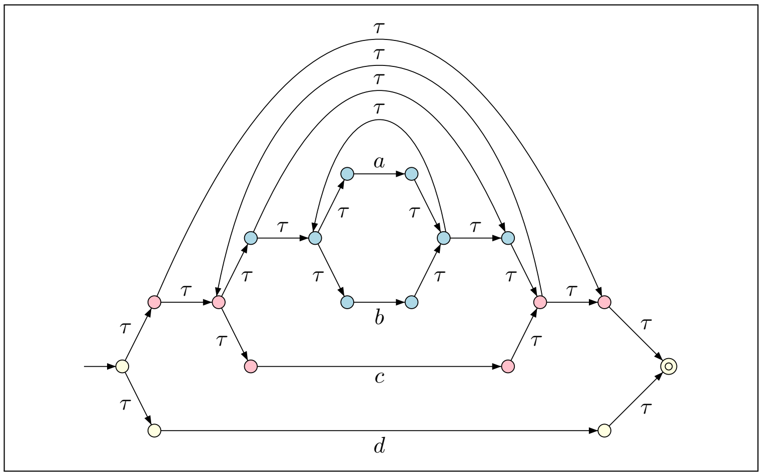
(b)



(c)



(d)



2.4 Properties of the class of regular languages

In this section we formally relate the concepts of a DFA, an NFA and a regular expression. We prove that they accept the same class of languages, the class of regular languages. We also provide a means to prove that a language is not in this class, i.e. that it is not regular. Moreover, we look at closure properties and decision procedures for the class of regular languages.

Definition 2.35. A language $L \subseteq \Sigma^*$ over an alphabet Σ is called a regular language if there exists an NFA N such that $L = \mathcal{L}(N)$.

The next theorem states that NFAs, DFAs and regular expressions define the same class of languages, viz. the class of regular languages. The theorem, or rather the theorems its proof refers to, also provides flexibility in the representation of a regular language. Given a regular language L as either the language accepted by an NFA or by a DFA or as the language of a regular expression, we can construct an NFA or a DFA that accepts L or a regular expression who's language is L.

Theorem 2.36. Let L be a language. The following three statements are equivalent:

(i) L is a regular language

(ii) There exists a DFA D such that $L = \mathcal{L}(D)$.

(iii) There exists a regular expression r such that $L = \mathcal{L}(r)$.

Proof. $[(i) \Rightarrow (ii)]$ By definition there exists an NFA N such that $L = \mathcal{L}(N)$. By Theorem 2.19 there exists a DFA D such that $\mathcal{L}(N) = \mathcal{L}(D)$. Then, clearly, $L = \mathcal{L}(D)$ for the DFA D.

$[(ii) \Rightarrow (iii)]$ Suppose $L = \mathcal{L}(D)$ for a DFA D. By Theorem 2.31 there exists a regular expresion r such that $\mathcal{L}(D) = \mathcal{L}(r)$. Thus clearly, $L = \mathcal{L}(r)$ for the regular expression r.

[$(iii) \Rightarrow (i)$] Suppose $L = \mathcal{L}(r)$ for a regular expression r. By Theorem 2.33 there exists an NFA N such that $\mathcal{L}(N) = \mathcal{L}(r)$. Therefore $L = \mathcal{L}(N)$ and L is a regular language. $\square$

Next we investigate closure properties of the class of regular languages. The following theorem states that the class of regular languages is closed under union, complement and intersection. We use the flexibility provided by Theorem 2.36 to choose or use the representation of a regular language that suits best.

Theorem 2.37.

(a) If L_1, L_2 are regular languages, then $L_1 \cup L_2$ is a regular language too.

(b) If the language $L \subseteq \Sigma^*$ is regular, then so is $L^C = \Sigma^* \setminus L$.

(c) If L_1, L_2 are regular languages, then $L_1 \cap L_2$ is a regular language too.

Proof. (a) By Theorem 2.36 we can find regular expressions r_1 and r_2 such that $L_1 = \mathcal{L}(r_1)$ and $L_2 = \mathcal{L}(r_2)$. Then we have $L_1 \cup L_2 = \mathcal{L}(r_1) \cup \mathcal{L}(r_2) = \mathcal{L}(r_1 + r_2)$. Thus, by Theorem 2.36 again, $L_1 \cup L_2$ is a regular language.

(b) Let, applying Theorem 2.36, $D = (Q, \Sigma, \delta, q_0, F)$ be a DFA such that $\mathcal{L}(D) = L$. Define the DFA $D' = (Q, \Sigma, \delta, q_0, F')$ by putting $F' = Q \setminus F$. Thus $q \in Q$ is a final state in D' iff q is not a final state in D. By definition we have $w \in \mathcal{L}(D')$ if both $(q_0, w) \vdash_{D'}^* (q, \varepsilon)$ and $q \in F'$. This is equivalent to $(q_0, w) \vdash_{D}^* (q, \varepsilon)$ and $q \notin F$, by definition of F'. But, this is exactly when $w \notin \mathcal{L}(D)$, since D is deterministic. (See Lemma 2.5.)

(c) By the laws of De Morgan, $L_1 \cap L_2 = (L_1^C \cup L_2^C)^C$. The languages L_1^C and L_2^C are regular, by regularity of L_1 and L_2 and part (b). Thus $L_1^C \cup L_2^C$ is regular, by part (a). Therefore, $(L_1^C \cup L_2^C)^C$ is regular, again by part (b). $\square$

Note, for item (b), it is important that the automaton we consider is deterministic. Changing acceptance and non-acceptance in an NFA does not lead in general to the complement of the accepted language.

A proof for part (a) and (c) of the theorem based on the construction of an automaton, as the proof for part (b), is possible as well. For this we make use of the so-called product automaton of two DFA. We first consider the case of the union $L_1 \cup L_2$ of two regular languages L_1 and L_2 over some alphabet Σ.

Suppose the DFA D_1 and D_2, with $D_i = (Q_i, \Sigma, \delta_i, q_0^i, F_i)$ for $i = 1, 2$, accept L_1 and L_2, respectively. We define the product DFA $D = (Q, \Sigma, \delta, q_0, F)$ as follows:

(i) The set of states Q is the Cartesian product $Q_1 \times Q_2$ of Q_1 and Q_2. Thus a state of Q is a pair of states $\langle q_1, q_2 \rangle$ with $q_1 \in Q_1$, $q_2 \in Q_2$.

(ii) The initial state of Q is therefore also a pair, viz. $\langle q_0^1, q_0^2 \rangle$, consisting of the initial state q_0^1 of Q_1, and the initial state q_0^2 of Q_2.

(iii) The transition function $\delta : (Q_1 \times Q_2) \times \Sigma \to Q_1 \times Q_2$ is the 'product' of the transition functions of D_1 and D_2. We put

$$\delta(\langle q_1, q_2 \rangle, a) = \langle q_1', q_2' \rangle \iff \delta_1(q_1, a) = q_1' \,\wedge\, \delta_2(q_2, a) = q_2'$$

(iv) The set of final states F comprises, in the case of the union, all pairs in Q of which at least one component is a final state, for D_1, for D_2, or for both. Thus $F = F_1 \times Q_2 \cup Q_1 \times F_2$. Put differently,

$$F = \{\, \langle q_1, q_2 \rangle \in Q \mid q_1 \in F_1 \,\vee\, q_2 \in F_2 \,\}$$

Next, we verify that D accepts $L_1 \cup L_2$. For this we make the following claim, which can be proven by induction on the length of w. *Claim*: for $q_1, q_1' \in Q_1$, $q_2, q_2' \in Q_2$, $w \in \Sigma^*$ it holds that

$$(q_1, w) \vdash_1^* (q_1', \varepsilon) \,\wedge\, (q_2, w) \vdash_1^* (q_2', \varepsilon) \iff (\langle q_1, q_2 \rangle, w) \vdash_D^* (\langle q_1', q_2' \rangle, w') \tag{2.2}$$

Using the claim we derive, for $w \in \Sigma^*$,

$$\begin{aligned}
& w \in \mathcal{L}(D_1) \cup \mathcal{L}(D_2) \\
& \quad \Leftrightarrow \quad \exists q_1 \in F_1 \colon (q_0^1, w) \vdash_1^* (q_1, \varepsilon) \,\vee\, \exists q_2 \in F_2 \colon (q_0^2, w) \vdash_2^* (q_2, \varepsilon) \\
& \quad \Leftrightarrow \quad \exists q_1 \in F_1,\, q_2 \in Q_2 \colon (q_0^1, w) \vdash_1^* (q_1, \varepsilon) \,\wedge\, (q_0^2, w) \vdash_2^* (q_2, \varepsilon) \,\vee \\
& \qquad\quad\; \exists q_1 \in Q_1,\, q_2 \in F_2 \colon (q_0^1, w) \vdash_1^* (q_1, \varepsilon) \,\wedge\, (q_0^2, w) \vdash_2^* (q_2, \varepsilon) \\
& \quad \Leftrightarrow \quad \exists q_1 \in F_1,\, q_2 \in Q_2 \colon (\langle q_0^1, q_0^2 \rangle, w) \vdash_D^* (\langle q_1, q_2 \rangle, \varepsilon) \,\vee \\
& \qquad\quad\; \exists q_1 \in Q_1,\, q_2 \in F_2 \colon (\langle q_0^1, q_0^2 \rangle, w) \vdash_D^* (\langle q_1, q_2 \rangle, \varepsilon) \\
& \quad \Leftrightarrow \quad \exists q \in F \colon (q_0, w) \vdash_D^* (q, \varepsilon) \quad \text{(by definition of } F\text{)} \\
& \quad \Leftrightarrow \quad w \in \mathcal{L}(D)
\end{aligned}$$

Thus $L_1 \cup L_2 = \mathcal{L}(D_1) \cup \mathcal{L}(D_2) = \mathcal{L}(D)$. Since $L_1 \cup L_2$ is a language accepted by a DFA, it is a language accepted by an NFA by Theorem 2.36. Hence, $L_1 \cup L_2$ is regular.

In order to show that the intersection $L_1 \cap L_2$ of two regular langauges is regular too, we can exploit the product automaton again without only a slight adaptation of its set of final states. Assume that the DFA D_1 and D_2, with $D_i = (\, Q_i,\, \Sigma,\, \delta_i,\, q_0^i,\, F_i \,)$ for $i = 1, 2$, accept L_1 and L_2, respectively. Define the product DFA $D = (\, Q,\, \Sigma,\, \delta,\, q_0,\, F \,)$ where Q, δ, and q_0 are as before, and now $F = F_1 \times F_2$. Thus, a state $\langle q_1, q_2 \rangle \in Q_1 \times Q_2$ is final iff both $q_1 \in Q_1$ and $q_2 \in Q_2$ are final, i.e. $q_1 \in F_1$ and $q_2 \in F_2$.

The proof that $\mathcal{L}(D) = \mathcal{L}(D_1) \cap \mathcal{L}(D_2)$ makes use of the claim of Equation (2.2) too. Now we argue

$$\begin{aligned}
& w \in \mathcal{L}(D_1) \cap \mathcal{L}(D_2) \\
& \quad \Leftrightarrow \quad \exists q_1 \in F_1 \colon (q_0^1, w) \vdash_1^* (q_1, \varepsilon) \,\wedge\, \exists q_2 \in F_2 \colon (q_0^2, w) \vdash_2^* (q_2, \varepsilon) \\
& \quad \Leftrightarrow \quad \exists q_1 \in F_1,\, q_2 \in F_2 \colon (\langle q_0^1, q_0^2 \rangle, w) \vdash_D^* (\langle q_1, q_2 \rangle, \varepsilon) \\
& \quad \Leftrightarrow \quad \exists q \in F \colon (q_0, w) \vdash_D^* (q, \varepsilon) \quad \text{(by the current definition of } F\text{)} \\
& \quad \Leftrightarrow \quad w \in \mathcal{L}(D)
\end{aligned}$$

We see that $L_1 \cap L_2 = \mathcal{L}(D_1) \cap \mathcal{L}(D_2) = \mathcal{L}(D)$. Since $L_1 \cap L_2$ is a language accepted by a DFA, it is a language accepted by an NFA because of Theorem 2.36. Therefore we conclude that $L_1 \cap L_2$ is a regular language.

In order to show that a language is a regular language, we can either show that it is the language of a regular expression, that is the language accepted by a DFA, or that it it the language accepted by an NFA. However, not every language is a regular language. But, so far we do not have means to acutally show that a language *isn't* regular. The next theorem, aptly called the Pumping Lemma, provides a tool to do so.

Theorem 2.38 (Pumping Lemma for regular languages)**.** Let L be a regular language over an alphabet Σ. There exists a constant $m > 0$ such that each $w \in L$ with $|w| \geqslant m$ can be written as $w = xyz$ where $x, y, z \in \Sigma^*$ are strings such that $y \neq \varepsilon$, $|xy| \leqslant m$, and for all $k \geqslant 0$: $xy^kz \in L$.

Proof. Suppose, with appeal to Theorem 2.36, $L = \mathcal{L}(D)$ for a DFA D. Choose m to be the number of states of D. Suppose $w \in L$ and $|w| \geqslant m$. Say $w = a_1 \cdots a_n$, thus $n \geqslant m$. Pick $n+1$ states $q_0, \ldots, q_n$ with q_0 the initial state of D, $\delta_D(q_{i-1}, a_i) = q_i$, for $1 \leqslant i \leqslant n$, and q_n a final state. Thus $(q_{i-1}, a_i a_{i+1} \cdots a_n) \vdash_D (q_i, a_{i+1} \cdots a_n)$. Since D has m states, the first $m+1$ states $q_0, \ldots, q_m$ cannot all be different. Pick m_1, m_2 such that $0 \leqslant m_1 < m_2 \leqslant m$ and $q_{m_1} = q_{m_2}$. Put

$$x = a_1 \cdots a_{m_1}, \qquad y = a_{m_1+1} \cdots a_{m_2}, \quad \text{and} \quad z = a_{m_2+1} \cdots a_n$$

We have $xyz = a_1 \cdots a_{m_1} a_{m_1+1} \cdots a_{m_2} a_{m_2+1} \cdots a_n = w$, $y \neq \varepsilon$ since $m_1 < m_2$, and $|xy| \leqslant m$ since $m_2 \leqslant m$.

We verify that $xy^kz \in L$ for all $k \geqslant 0$: It holds that

$$(q_0, x) \vdash_D^* (q_{m_1}, \varepsilon), \qquad (q_{m_1}, y) \vdash_D^* (q_{m_2}, \varepsilon), \quad \text{and} \quad (q_{m_2}, z) \vdash_D^* (q_n, \varepsilon)$$

Since $q_{m_1} = q_{m_2}$, it follows that $(q_{m_1}, y) \vdash_D^* (q_{m_1}, \varepsilon)$. Thus $(q_{m_1}, y^k) \vdash_D^* (q_{m_1}, \varepsilon)$ (by Lemma 2.5 and an inductive argument), and $(q_{m_1}, y^k) \vdash_D^* (q_{m_2}, \varepsilon)$ for arbitrary $k \geqslant 0$. Therefore

$$(q_0, x) \vdash_D^* (q_{m_1}, \varepsilon), \qquad (q_{m_1}, y^k) \vdash_D^* (q_{m_2}, \varepsilon), \quad \text{and} \quad (q_{m_2}, z) \vdash_D^* (q_n, \varepsilon)$$

for arbitrary $k \geqslant 0$, and hence $(q_0, xy^kz) \vdash_D^* (q_n, \varepsilon)$. Thus $w = xy^kz \in \mathcal{L}(D) = L$ for all $k \geqslant 0$, since q_n is a final state. □

Essential use is made of the fact that the DFA has finitely many states only: $m+1$ states are chosen, viz. $q_0, \ldots, q_m$, of which there are at most m states different. So, at least one state is doubled. This state is the begin and end point of the loop y that can be taken any number of times –zero, one ore more– depending on the input string.

Consider the DFA D given by Figure 2.18. We claim that a choice for $m = 3$ will satisfy the claim of the Pumping Lemma. A string w of 3 symbols or more symbols that is

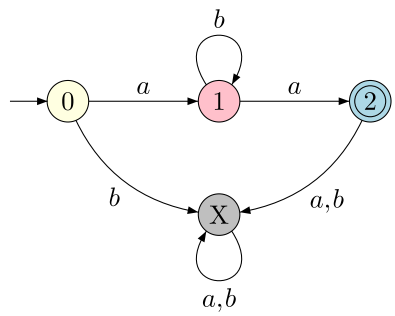


Figure 2.18: DFA hits the b-loop for input of sufficient length

accepted by D follows a path from the initial state q_0 to the final state q_2 visiting 4 or more states. There only 3 different states to consider as q_x is a sink state. So, one or more states are visited more often. In this particular case this is state q_1. It follows that the b-loop is done mulitple times, in fact $\ell = |w| - 2$ times. If we split up w in x, y and z, i.e. $w = xyz$, with $x = a$, $y = b$, and $z = b^{\ell-1}a$, we have $|xy| \leqslant 3$, $y \neq \varepsilon$ and $xy^kz = ab^kb^{\ell-1}a = ab^{k+\ell-1}a$ is accepted by D.

The Pumping Lemma for regular languages is mainly used to prove *negative* results, i.e. it is used to prove that a language is *not* regular. One can do so by exploiting the 'reverse' of the Pumping Lemma: for each $m > 0$ a string $w \in L$ is given for which no split up in x, y and z meeting the extra requirements in possible. In particular, a split-up of w as $w = xyz$ with $|xy| \leqslant m$ and $y \neq \varepsilon$ will give rise to a string $w' = xy^kz$, for some $k \geqslant 0$, which is *not* in L. By Theorem 2.38 it then follows that the language cannot be regular. We provide two examples of this technique.

Example 2.39. The language $L = \{\, a^nb^n \mid n \geqslant 0 \,\}$ is not a regular language. Let $m > 0$ be arbitrary. Consider the string $w = a^mb^m$. We have that $w \in L$. Suppose we split $w = xyz$ such that $|xy| \leqslant m$, and $y \neq \varepsilon$. Then the string y is a non-empty string of a's, say $y = a^\ell$. Thus the string $w' = xy^2z = a^{m+\ell}b^m$, hence $w' \notin L$. We conclude that there is no constant m as mentioned by the Pumping Lemma, and therefore L is not a regular language.

Example 2.40. The language $L = \{\, a^{n^2} \mid n \geqslant 0 \,\}$ is not a regular language. Choose any $m > 0$. Consider $w = a^{m^2} \in L$. Suppose we can split $w = xyz$ such that $|xy| \leqslant m$, and $y \neq \varepsilon$. Then the string y is a non-empty string of a's, say $y = a^\ell$ with $1 \leqslant \ell \leqslant m$. Put $w' = xy^2z$, then we have $w' = a^{m^2+\ell}$. But $m^2 < m^2 + \ell \leqslant m^2 + m < m^2 + 2m + 1 = (m+1)^2$, thus $m^2 + \ell$ isn't a square. So, $w' \notin L$. Thus, we conclude that there is no constant m as mentioned by the Pumping Lemma, and therefore L is not a regular language.

We close the chapter by looking into two decision algorithms for regular languages. The first decision algorithm needs to determine whether given a *regular* language L as input,

L is or is not the empty language. We first solve the question if L is given by a regular expression. With appeal to Theorem 2.36 we can conclude that the theorem holds as well if L is given as the language accepted by an NFA.

Theorem 2.41. Let L be a regular language over an alphabet Σ represented by an NFA N accepting L. Then it can be decided if $L = \emptyset$ or not.

Proof. We decide, for a regular expression r, emptiness of $L(r)$ as follows:

- $L = \emptyset$ if $r = \mathbf{0}$;
- $L \neq \emptyset$ if $r = \mathbf{1}$;
- $L \neq \emptyset$ if $r = a$ for some $a \in \Sigma$;
- $L = \emptyset$ if $r = r_1 + r_2$ for two regular expressions r_1 and r_2 and both $\mathcal{L}(r_1)$ and $\mathcal{L}(r_2)$ are empty, $L \neq \emptyset$ if $\mathcal{L}(r_1)$ or $\mathcal{L}(r_2)$ is non-empty;
- $L = \emptyset$ if $r = r_1 \cdot r_2$ for two regular expressions r_1 and r_2 and either $\mathcal{L}(r_1)$ or $\mathcal{L}(r_2)$ is empty, $L \neq \emptyset$ if both $\mathcal{L}(r_1)$ and $\mathcal{L}(r_2)$ are non-empty;
- $L \neq \emptyset$ if $r = (r')^*$ for some regular expression r' (since $\varepsilon \in (r')^*$ for every r').

Note that the decision procedure terminates since the recusive calls for r_1, r_2 and r' above involve a structurally simpler argument.

Now, suppose $L = \mathcal{L}(N)$ for an NFA N. Construct, using the algorithms given in the proofs of Theorem 2.19 and Theorem 2.31, a regular expression r such that $L = \mathcal{L}(r)$ and decide whether $\mathcal{L}(r) = \emptyset$. □

Finally we consider a decision algorithm for membership. Given an arbitrary regular language $L \subseteq \Sigma^*$ and a string $w \in \Sigma^*$, is it the case or not that $w \in L$?

Theorem 2.42. Let $L \subseteq \Sigma^*$ be a regular language over the alphabet Σ, represented by an NFA N accepting L, and let $w \in \Sigma^*$ be a string over Σ. Then it can be decided if $w \in L$ or not.

Proof. Construct, using the algorithm given in the proof of Theorem 2.19, a DFA D such that $\mathcal{L}(D) = \mathcal{L}(N)$. Simulate D starting from its initial state on input w, say $(q_0, w) \vdash_D^* (q', \varepsilon)$ for some state q' of D. Decide $w \in L$ if q' is a final state of D; decide $w \notin L$ otherwise. □

Exercises for Section 2.4

Exercise 2.4.21. Prove using the Pumping Lemma, Theorem 2.38, that the following languages are *not* regular.

(a) $L_1 = \{\, a^k b a^k \mid k \geqslant 0 \,\}$

(b) $L_2 = \{\, a^k b^\ell \mid k > \ell > 0 \,\}$

(c) $L_3 = \{\, a^k b^\ell c^{k+\ell} \mid k, \ell \geqslant 0 \,\}$

Answer to Exercise 2.4.21

(a) Let $m > 0$ be arbitrary. Consider the string $w = a^m b a^m$. We have $w \in L_1$. Suppose $w = xyz$ is a split-up of w with $|xy| \leqslant m$ and $y \neq \varepsilon$. Then x is an arbitrary string of a's, say $x = a^{\ell_1}$ with $0 \leqslant \ell_1 < m$, y is a non-empty string of a's, say $y = a^{\ell_2}$ with $0 < \ell_2 \leqslant m$, and $z = a^{m-\ell_1-\ell_2} b a^m$. Thus the string $w' = xy^2z$ looks like $a^{\ell_1} a^{\ell_2} a^{\ell_2} a^{m-\ell_1-\ell_2} b a^m = a^{\ell_1+\ell_2+\ell_2} a^{m-\ell_1-\ell_2} b a^m = a^{m+\ell_2} b a^m$. Hence $w' \notin L_1$ since $m + \ell_2 \neq m$ because $\ell_2 \neq 0$. We conclude that there is no constant m as mentioned by the Pumping Lemma, and therefore L_1 is not a regular language.

(b) This is a variant. Pick any $m > 0$. Consider the string $w = a^m b^{m-1}$. We have $w \in L_2$. Assume we can write $w = xyz$ for strings x, y and z such that $|xy| \leqslant m$ and $y \neq \varepsilon$. Then we have $x = a^{\ell_1}$ with $0 \leqslant \ell_1 < m$, $y = a^{\ell_2}$ with $0 < \ell_2 \leqslant m$, and $z = a^{m-(\ell_1+\ell_2)} b^{m-1}$. Consider the string $w' = xy^0z$, i.e. $w' = xz$ since $y^0 = \varepsilon$. Then we have $w' = a^{\ell_1} a^{m-(\ell_1+\ell_2)}$. But, since $\ell_1 + m - (\ell_1+\ell_2) \leqslant m-1$, $w' \notin L_2$. We conclude that, since no $m > 0$ exists meeting the requirements of the Pumping Lemma, L_2 is not a regular language.

(c) Let $m > 0$. Consider the string $w = a^m b^m c^{2m}$. Then $w \in L_3$. As usually, assume we can write $w = xyz$ for strings x, y and z such that $|xy| \leqslant m$ and $y \neq \varepsilon$. Then it must be the case that $x = a^{\ell_1}$ with $0 \leqslant \ell_1 < m$, $y = a^{\ell_2}$ with $0 < \ell_2 \leqslant m$, and $z = a^{m-(\ell_1+\ell_2)} b^m c^{2m}$. Consider the string $w' = xy^2z$. Then it holds that $w' = a^{\ell_1} a^{\ell_2} a^{\ell_2} a^{m-(\ell_1+\ell_2)} b^m c^{2m} = a^{m+\ell_2} b^m c^{2m}$. But, then $w' \notin L_3$ since $\ell_2 > 0$ and therefore $m + \ell_2 + m \neq 2m$.

Exercise 2.4.22. Prove that the language $L_4 = \{\, vv^R \mid v \in \{a,b\}^* \,\}$ is not regular.

Answer to Exercise 2.4.22 Let $m > 0$ be arbitrary. Consider the string $w = a^m bba^m$. We have $w \in L_4$. Suppose $w = xyz$ is a split-up of w with $|xy| \leqslant m$ and $y \neq \varepsilon$. Then $x = a^{\ell_1}$ with $0 \leqslant \ell_1 < m$, $y = a^{\ell_2}$ with $0 < \ell_2 \leqslant m$, and $z = a^{m-\ell_1-\ell_2} bba^m$. Now consider the string $w' = xy^2z$. It holds that $w' = a^{\ell_1} a^{\ell_2} a^{\ell_2} a^{m-\ell_1-\ell_2} bba^m = a^{\ell_1+\ell_2+\ell_2} a^{m-\ell_1-\ell_2} bba^m = a^{m+\ell_2} ba^m$. Since $\ell_2 \neq 0$ we have $m + \ell_2 \neq m$. So, bb is not in the middle of the string w', hence w' is not of the form vv^R. We conclude that there is no constant m as mentioned by the Pumping Lemma, and therefore L_4 is not a regular language.

Exercise 2.4.23. Prove that the language $L_5 = \{\, a^n \mid n \text{ is prime} \,\}$ is not regular.

Answer to Exercise 2.4.23 Let $m > 0$ be arbitrary. Pick a prime number p such that $p \geqslant m$. Consider the string $w = a^p$. We have $w \in L_5$. Suppose $w = xyz$ is a split-up of w with $|xy| \leqslant m$ and $y \neq \varepsilon$. Then $x = a^{\ell_1}$ with $0 \leqslant \ell_1 < m$, $y = a^{\ell_2}$ with

$0 < \ell_2 \leqslant m$, and $z = a^{p-(\ell_1+\ell_2)}$. Now consider the string $w' = xy^{p+1}z$. It holds that $w' = a^{\ell_1+(p+1)\cdot\ell_2+p-(\ell_1+\ell_2)} = a^{p+p\cdot\ell_2}$. Since $\ell_2 \neq 0$, the number $p+p\cdot\ell_2$ is not prime, for $p+p\cdot\ell_2 = p\cdot(1+\ell_2)$. So, there is no constant m as mentioned by the Pumping Lemma, and therefore L_5 is not a regular language.

Exercise 2.4.24.

(a) Prove, by induction on m, that $m < 2^m$ for $m \geqslant 0$.

(b) Prove that the language $L_6 = \{\, a^n \mid n = 2^k \text{ for some } k \geqslant 0 \,\}$ is not regular.

Answer to Exercise 2.4.24

(a) Basis, $m = 0$: Clear, we have $2^0 = 1$ and $0 < 1$. Induction step, $m+1$: By induction hypothesis $m < 2^m$. Thus $m + 1 \leqslant m + m < 2^m + 2^m = 2^{m+1}$.

(b) Choose any $m > 0$. Consider the string $w = a^{2^m}$. Then $w \in L_6$. Assume that there is a split-up $w = xyz$ with $|xy| \leqslant m$ and $y \neq \varepsilon$. Then $x = a^{\ell_1}$ with $0 \leqslant \ell_1 < m$, $y = a^{\ell_2}$ with $0 < \ell_2 \leqslant m$, and $z = a^{m-(\ell_1+\ell_2)}$. Put $w' = xy^2z$. Then $w' = a^{2^m+\ell_2}$. However, $2^m < 2^m + \ell_2$ since $\ell_2 > 0$, while $2^m + \ell_2 \leqslant 2^m + m < 2^m + 2^m = 2^{m+1}$ by part (a). Therefore, $2^m + \ell_2$ is *not* a power of 2 and $w' \notin L_6$. So, we cannot find $m > 0$ that satisfies the conditions of the Pumping Lemma, and therefore L_6 is not regular.

Exercise 2.4.25. Prove that the following languages are not regular.

(a) $L_7 = \{\, w \in \{a,b\}^* \mid \#_a(w) = \#_b(w) \,\}$

(b) $L_8 = \{\, w \in \{a,b\}^* \mid \#_a(w) \neq \#_b(w) \,\}$

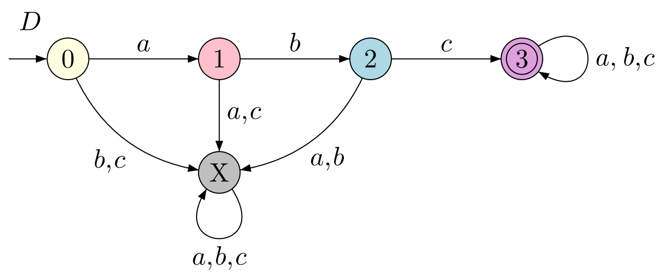
Answer to Exercise 2.4.25

(a) Choose any $m > 0$. Consider the string $w = a^m b^m$. Clearly $\#_a(w) = \#_b(w)$, thus $w \in L_7$. Assume $w = xyz$ for strings x, y and z with $|xy| \leqslant m$ and $y \neq \varepsilon$. Then $x = a^{\ell_1}$ with $0 \leqslant \ell_1 < m$, $y = a^{\ell_2}$ with $0 < \ell_2 \leqslant m$, and $z = a^{m-(\ell_1+\ell_2)}b^m$. Put $w' = xy^2z$. Then $w' = a^{m+\ell_2}b^m$. Since $\#_a(w') = \#_b(w')$, we have $w' \notin L_7$. So, no $m > 0$ has the properties as guaranteed by the Pumping Lemma for a regular language. Therefore L_7 is not regular.

(b) Applying the Pumping Lemma directly doesn't work. Instead we make use of a closure property of the class of regular languages. According to Theorem 2.37b the complement of a regular language is a regular language itself. Now, the complement of the language L_8 is precisely the language L_7. But, according to part (a), L_7 isn't a regular language. So, L_8 is neither.

Exercise 2.4.26. For a string $w \in \Sigma^*$ for some alphabet Σ, the reversal $w^R \in \Sigma^*$ of w is defined as follows: (i) $\varepsilon^R = \varepsilon$, (ii) $(av)^R = v^R a$.

(a) Consider the DFA D that accepts the language $L_9 = \{ abcw \mid w \in \{a,b,c\}^* \}$.



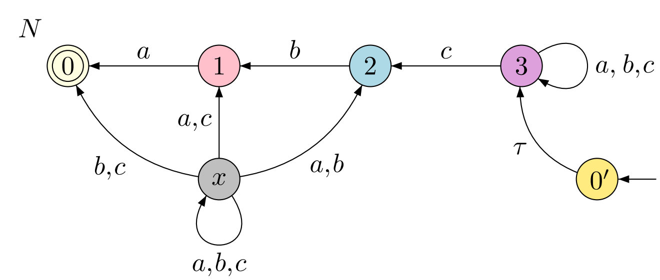
Construct, using the the DFA D, an NFA N that accepts

$$L^R = \{ wcba \mid w \in \{a,b,c\}^* \}$$

(b) Prove that the class of regular languages is closed under reversal, i.e. if a language L is regular, then so is $L^R = \{ w^R \mid w \in L \}$.

Answer to Exercise 2.4.26

(a) We add a new initial state q_0' that is connected to the final state q_3 of D, which is not final in N. The only final state of N is q_0. All arrows of D are reversed in N. Note multiple outgoing transitions for various states on various symbols. Also note, the trap state q_x of D is not reachable in N, and can be left out.



(b) Suppose, in view of Theorem 2.36, that L is the language accepted by the DFA $D = (\, Q,\, \Sigma,\, \delta_D,\, q_0,\, F \,)$. Define the NFA $N = (\, Q',\, \Sigma,\, \rightarrow_N,\, q_0',\, \{q_0\} \,)$ as follows: Let q_0' be a fresh state not in Q. We put $Q' = Q \cup \{q_0'\}$. The transition relation $\rightarrow_N$ satisfies

$$\forall q \in Q\colon \quad q_0' \xrightarrow{\tau}_N q \quad \text{if } q \in F$$

$$\forall q, q' \in Q\, \forall a \in \Sigma\colon \quad q' \xrightarrow{a}_N q \quad \text{if } \delta_D(q,a) = q'$$

One can prove, exploiting the fact that N has no τ-transitions for states $q \in Q$, that

$$(q, w) \vdash_D^* (q', \varepsilon) \iff (q', w^R) \vdash_N^* (q, \varepsilon) \tag{2.3}$$

From this it follows that

$$\begin{aligned} w \in \mathcal{L}(D) \;&\Leftrightarrow\; \exists q \in F \colon (q_0, w) \vdash_D^* (q, \varepsilon) \\ &\Leftrightarrow\; \exists q \in F \colon (q, w^R) \vdash_N^* (q_0, \varepsilon) \;\Leftrightarrow\; w^R \in \mathcal{L}(N) \end{aligned}$$

Thus $\mathcal{L}(N) = \mathcal{L}(D)^R = L^R$ and L^R is a regular language.

Exercise 2.4.27. The symmetric difference $X \mathbin{\triangle} Y$ of two sets X and Y is given by

$$X \mathbin{\triangle} Y = \{\, x \in X \mid x \notin Y \,\} \cup \{\, y \in Y \mid y \notin X \,\}$$

Prove that the class of regular languages is closed under symmetric difference, i.e. if the languages L_1 and L_2 are regular, then so is $L_1 \mathbin{\triangle} L_2$.

Answer to Exercise 2.4.27 We construct variant of the product automaton that accepts $L_1 \mathbin{\triangle} L_2$. Suppose the DFA D_1 and D_2 over the alphabet Σ, with $D_i = (\, Q_i, \Sigma, \delta_i, q_0^i, F_i \,)$ for $i = 1, 2$, accept L_1 and L_2, respectively. Define the product DFA $D = (\, Q, \Sigma, \delta, q_0, F \,)$ where Q, δ, and q_0 are as before:

(i) $Q = Q_1 \times Q_2$;

(ii) $\delta(\langle q_1, q_2 \rangle, a) = \langle q_1', q_2' \rangle$ if $\delta_1(q_1, a) = q_1'$ and $\delta_2(q_2, a) = q_2'$;

(iii) $q_0 = \langle q_0^1, q_0^2 \rangle$

Now we define the set of states $F = (F_1 \times (Q_2 \setminus F_2)) \cup (Q_1 \setminus F_1) \times F_2$, i.e.

$$F = \{\, \langle q_1, q_2 \rangle \in Q_1 \times Q_2 \mid (q_1 \in F_1 \wedge q_2 \notin F_2) \vee (q_1 \notin F_1 \wedge q_2 \in F_2) \,\}$$

To prove that $\mathcal{L}(D) = L_1 \mathbin{\triangle} L_2$ we first argue

$$\begin{aligned} & w \in \mathcal{L}(D_1) \mathbin{\triangle} \mathcal{L}(D_2) \\ &\quad\Leftrightarrow\; w \in \mathcal{L}(D_1) \setminus \mathcal{L}(D_2) \vee w \in \mathcal{L}(D_2) \setminus \mathcal{L}(D_1) \\ &\quad\Leftrightarrow\; \exists q_1 \in Q_1,\; q_2 \in Q_2 \colon (q_0^1, w) \vdash_1^* (q_1, \varepsilon) \wedge (q_0^2, w) \vdash_2^* (q_2, \varepsilon) \wedge \\ &\qquad\qquad \big((q_1 \in F_1 \wedge q_2 \notin F_2) \vee (q_1 \notin F_1 \wedge q_2 \in F_2)\big) \\ &\quad\Leftrightarrow\; \exists q_1 \in Q_1,\, q_2 \in Q_2 \colon (\langle q_0^1, q_0^2 \rangle, w) \vdash_D^* (\langle q_1, q_2 \rangle, \varepsilon) \wedge \langle q_1, q_2 \rangle \in F \\ &\quad\Leftrightarrow\; \exists q \in F \colon (q_0, w) \vdash_D^* (q, \varepsilon) \\ &\quad\Leftrightarrow\; w \in \mathcal{L}(D) \end{aligned}$$

Thus $L_1 \mathbin{\triangle} L_2 = \mathcal{L}(D_1) \mathbin{\triangle} \mathcal{L}(D_2) = \mathcal{L}(D)$ is accepted by a DFA. By Theorem 2.36 is follows that $L_1 \mathbin{\triangle} L_2$ is regular.

2.5 Constructing a minimal DFA

For a regular language L there are many DFA that accept L. In this section we look for a minimal DFA, i.e. a DFA that accepts L while no other DFA with fewer states that does this. We first introduce the notion of L-equivalence to identify or distinguish states in a DFA, initial or not, that accept the same part of L. States that are identified by L-equivalence can be taken together. Starting from the assumption that it is known which states of the given DFA are L-equivalent, the minimal DFA can be obtained via a quotient construction. We will proves that this gives a minimal DFA indeed. In addition, we give an algorithm, and a proof of its correctness, to find L-equivalent states of the DFA started from.

We start off with the central notion of this section, viz. L-equivalence for the states of a DFA accepting the language L.

Definition 2.43. Let D be a DFA with set of states Q, set of final states F, and accepted language L. Two states q_1, q_2 of D are called L-equivalent, notation $q_1 \approx_L q_2$, if

$$\delta(q_1, w) \in F \iff \delta(q_2, w) \in F$$

for all strings $w \in \Sigma^*$.

From the definition it follows that $\delta(q_1, a) \approx_L \delta(q_2, a)$ if $q_1 \approx_L q_2$, for all $a \in \Sigma$. This can be seen as follows: Put $q_1' = \delta(q_1, a)$ and $q_2' = \delta(q_2, a)$. Then, for all $w \in \Sigma^*$, $\delta(q_1', w) \in F$ iff $\delta(q_1, aw) \in F$ iff $\delta(q_2, aw) \in F$ iff $\delta(q_2', w) \in F$.

Example 2.44. Consider the DFA D given by Figure 2.19. Put $L = \mathcal{L}(D)$. The states q_0 and q_2 of this automaton are L-equivalent: $\delta(q_0, w) \in F$ iff $w \in (bb)^* + (bb)^* {\cdot} aa {\cdot} (aa+bb)^*$ iff $w \in (aa + bb)^*$, and $\delta(q_2, w) \in F$ iff $w \in (aa + bb)^*$.

It follows that the states q_4 and q_6 are L-equivalent too, since (i) $q_4, q_6 \notin F$, (ii) $\delta(q_4, aw) \in F$ iff $\delta(q_5, w) \in F$ iff $\delta(q_6, aw) \in L$, and (iii) $\delta(q_4, bw) \in F$ iff $\delta(q_0, w) \in F$ iff $\delta(q_2, w) \in F$ iff $\delta(q_6, bw) \in F$.

The state q_1 is not L-equivalent to the states q_0 and q_2. The latter are final states, thus $\delta(q_0, \varepsilon), \delta(q_2, \varepsilon) \in F$, but the former is not a final state, thus $\delta(q_1, \varepsilon) \notin F$. Also, q_1 is not L-equivalent to any of q_4, q_5 and q_7. For example $\delta(q_1, a) \in F$, which does not hold for the other states mentioned.

Finally, since state q_5 is a trap state, we have $\delta(q_5, w) \notin F$, for all $w \in \Sigma^*$. For state q_3 is holds that $\delta(q_3, w) \in F$ iff $w \in b^* {\cdot} a$, and for state q_7, $\delta(q_7, w) \in F$ iff $w \in a {\cdot} b^* {\cdot} a$. It follows that q_3 is not equivalent to any other state, and so are q_5 and q_7. Note, states q_3 and q_7 are not reachable from the initial state q_0.

It is straightforward to verify that for a DFA D, the relation $\approx_L$, for $L = \mathcal{L}(D)$, is an equivalence relation. We write $[q]_L = \{ q' \in Q \mid q \approx_L q' \}$ to denote the equivalence class of $\approx_L$ containing the state $q \in Q$. Similarly, we put $[Q']_L = \bigcup_{q' \in Q'} [q']_L$ for the union of the equivalence classes of elements of a subset of states $Q' \subseteq Q$.

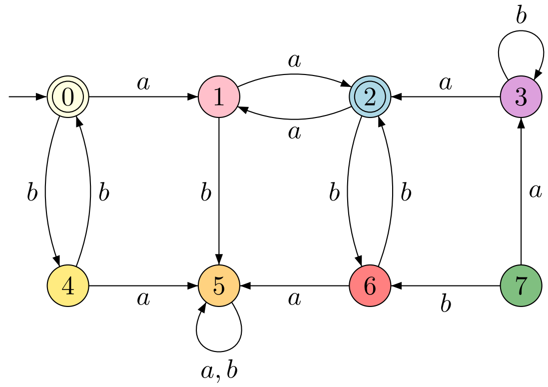


Figure 2.19: L-equivalent states q_0, q_2 and q_4, q_6

Definition 2.45. Let $D = (Q, \Sigma, \delta, q_0, F)$ be a DFA with accepted language L. The DFA $D_L = (Q_L, \Sigma, \delta_L, C_0, F_L)$, called the quotient DFA of D, has equivalence classes of $\approx_L$ as its states, i.e. $Q_L = Q/{\approx_L}$, the same alphabet Σ as D has, the equivalence class $[q_0]_L$ of the initial state of D as its own initial state, i.e. $C_0 = [q_0]_L$, the equivalence classes of D's final states as its own final states, i.e. $F_L = \{ [q]_L \mid q \in F \}$, and has a transition function δ_L defined by $\delta_L([q]_L, a) = [\delta(q, a)]_L$, for $q \in Q$, $a \in \Sigma$.

We need to verify that the function $\delta_L : Q_L \times \Sigma \to Q_L$ is well-defined: if $q_1 \approx_L q_2$, then $\delta(q_1, a) \approx_L \delta(q_2, a)$. Because then $[\delta(q_1, a)]_L = [\delta(q_2, a)]_L$, and in Definition 2.45 above, the outcome of $\delta_L([q]_L, a)$ is independent of the representative q. So, choose $q_1, q_2 \in Q$ such that $q_1 \approx_L q_2$, and pick a symbol $a \in \Sigma$. Put $q_1' = \delta(q_1, a)$, $q_2' = \delta(q_2, a)$. We claim that $q_1' \approx_L q_2'$: for any $w \in \Sigma^*$ it holds that $\delta(q_1', w) \in F$ iff $\delta(q_1, aw) \in F$ iff $\delta(q_2, aw) \in F$ iff $\delta(q_2', w) \in F$. Thus $q_1' \approx_L q_2'$, i.e. $\delta(q_1, a) \approx_L \delta(q_2, a)$, as was to be shown.

Example 2.46. Returning to the DFA D of Figure 2.19, we distinguish the L-equivalence classes $\{q_0, q_2\}$, $\{q_1\}$, $\{q_4, q_6\}$ and $\{q_5\}$, as well as $\{q_3\}$ and $\{q_7\}$. Following Definition 2.45, we obtain a DFA D_L as depicted in Figure 2.20: the states are the equivalence classes of $\approx_L$, the equivalence class $\{q_0, q_2\}$ which contains q_0 is the initial state, final state is the equivalence class $\{q_0, q_2\}$ too. The transitions are inherited from the DFA D. E.g., $\delta_L(\{q_0, q_2\}, a) = \{q_1\}$ since both $\delta(q_0, a) = q_1$, and $\delta(q_2, a) = q_1$. Likewise, $\delta_L(\{q_0, q_2\}) = \{q_4, q_6\}$ since $\delta(q_0, a) = q_4$ and $\delta(q_2, a) = q_6$. Also, $\delta_L(\{q_3\}) = \{q_0, q_2\}$ since $\delta(q_3, a) = q_2$. Note, we do not have $\delta(q_3, a) = q_0$.

If we remove from the quotient DFA D_L of Figure 2.20 the non-reachable states $\{q_3\}$ and $\{q_7\}$ we obtain a minimal representation of the DFA D of Figure 2.19. See Figure 2.21. Off course, we could have better started from a DFA, smaller than D, having reachable states only, since this reduces the number of L-equivalences to check. Intuitively, after the superfluous states q_3 and q_7 are dispensed with, the states q_2 and q_6 are

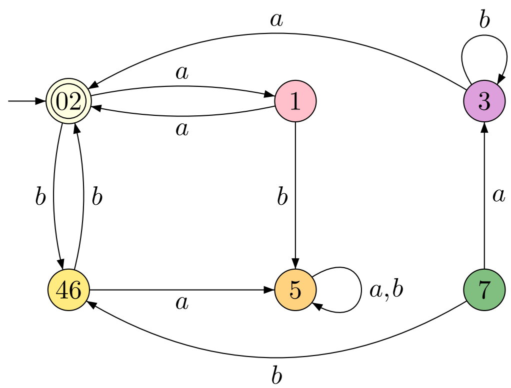


Figure 2.20: The quotient DFA D_L

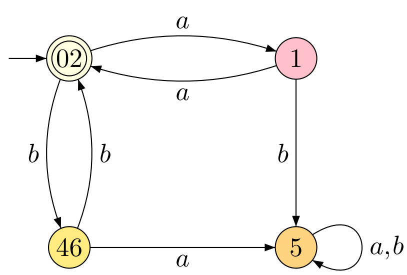


Figure 2.21: A minimized DFA

folded along the line through states q_1 and q_5 onto the states q_0 and q_4, respectively. Note that this way the transitions involved, as well states being final or not, are preserved.

Clearly, the DFA of Figure 2.21 also accepts the language $(aa+bb)^*$ and has 4 states only. The construction of (i) restricting to reachable states, and (ii) taking a quotient modulo $\approx_L$, indeed provides a DFA accepting the same language as the original DFA. Theorem 2.48 below claims that this is the case generally. In addition, the theorem states that the DFA obtained this way is of minimal size. For the proof of the theorem we need an auxiliary result.

Lemma 2.47. Let the DFA D and D_L be as given by Definition 2.45. For all states $q \in Q$ and all states $C \in Q_L$ such that $q \in C$, it holds that

$$\delta(q,w) \in F \iff \delta_L(C,w) \in F_L$$

for all strings $w \in \Sigma^*$.

Proof. Assume $q \in C$ for a state q of D and a state C of D_L.

($\Rightarrow$) Induction on w. Basis, $w = \varepsilon$: Suppose $\delta(q,\varepsilon) \in F$. Then $q \in F$. So, $C \in F_L$ by definition of F_L. Thus, $\delta_L(C,\varepsilon) \in F_L$. Induction step, $w = aw'$: Suppose

$\delta(q, a) = q'$. Put $C' = [q']_L$. Then $\delta_L(C, a) = C'$ by definition of δ_L. Also $q' \in C'$. By induction hypothesis, $\delta(q', w') \in F$ implies $\delta_L(C', w') \in F_L$. Suppose, $\delta(q, w) \in F$. Then $\delta(q', w') \in F$. Thus $\delta_L(C', w') \in F_L$, and $\delta_L(C, w) \in F_L$.

($\Leftarrow$) Induction on w. Basis, $w = \varepsilon$: Suppose $\delta_L(C, \varepsilon) \in F_L$, i.e. $C \in F_L$. Choose $\bar{q} \in C \cap F$. Since $q, \bar{q} \in C$, we have $q \approx_L \bar{q}$. In particular, $\delta(q, \varepsilon) \in F$ iff $\delta(\bar{q}, \varepsilon) \in F$. Since $\bar{q} \in F$, it follows that $q \in F$. Induction step, $w = aw'$: Put $C' = \delta_L(C, a)$. So, we can pick $\bar{q} \in C$, $\bar{q}' \in C'$ such that $\delta(\bar{q}, a) = \bar{q}'$. Put $q' = \delta(q, a)$. Since $q, \bar{q} \in C$, we have $q \approx_L \bar{q}$. Therefore, $\delta(q, a) \approx_L \delta(\bar{q}, a)$, i.e. $q' \approx_L \bar{q}'$. Since $\bar{q}' \in C'$, and C' is an equivalence class of $\approx_L$, it follows that $q' \in C'$. By induction hypothesis, $\delta_L(C', w') \in F_L$ iff $\delta(q', w') \in F$. Suppose, $\delta_L(C, w) \in F_L$. Then $\delta_L(C', w') \in F_L$. Thus $\delta(q', w') \in F$, and $\delta(q, w) \in F$. $\square$

Next we show the correctness of the quotient construction to find a minimal DFA D_L accepting the same language as a given DFA D. Initially we assume that D has only reachable states.

Theorem 2.48. Let D be a DFA with accepted language L. Assume that all states of D are reachable. Let D_L be the quotient DFA of D with set of states Q_L. Then $\mathcal{L}(D_L) = L$. Moreover, if for a DFA D' with set of states Q' it holds that $\mathcal{L}(D') = L$, then $|Q_L| \leqslant |Q'|$.

Proof. Let $D = (Q, \Sigma, \delta, q_0, F)$, and $D_L = (Q_L, \Sigma, \delta_L, C_0, F_L)$.

For the initial state q_0 of D and the initial state C_0 of D_L it holds that $q_0 \in C_0$. Application of Lemma 2.47 yields $\delta(q_0, w) \in F$ iff $\delta_L(C_0, w) \in F_L$, for all $w \in \Sigma^*$. In other words $\mathcal{L}(D) = \mathcal{L}(D_L)$.

Suppose $D' = (Q', \Sigma, \delta', q'_0, F')$ accepts L. We claim

$$\forall C \in Q_L\, \exists q' \in Q'\, \forall w \in \Sigma^* \colon \delta_L(C, w) \in F_L \iff \delta'(q', w') \in F'$$

Informally, for each state of D_L we can find an 'equivalent' state of D'. We want to prove the claim by induction. To this end we define the notion of the minimal path length $mp\ell(C)$ of a state C of D_L:

$$mp\ell(C) = \min\{\, n \in \mathbb{N} \mid \exists w \in \Sigma^n \colon \delta_L(C_0, w) = C \,\}$$

Since all states of D_L are reachable, this notion is well-defined. With this notion in place, we proceed proving the claim, by induction on the minimal path length $mp\ell(C)$ of a state C of D_L.

Basis, $mp\ell(C) = 0$: It holds that $C = C_0$. Since $\mathcal{L}(D_L) = \mathcal{L}(D)$, as shown above, and, by assumption $\mathcal{L}(D') = \mathcal{L}(D)$, we have $\mathcal{L}(D_L) = \mathcal{L}(D')$. Thus $\delta_L(C_0, w) \in F_L$ iff $\delta'(q'_0, w) \in F'$. So, we can pick q'_0 to correspond to C_0. Induction step, $mp\ell(C) = n+1$: Suppose $C = \delta_L(C_0, va)$ for a string $v \in \Sigma^n$ and a symbol $a \in \Sigma$. Put $\bar{C} = \delta_L(C_0, v)$. Then $mp\ell(\bar{C}) = n$ and $\delta_L(\bar{C}, a) = C$. By induction hypothesis we can choose a state $\bar{q}' \in Q'$ such that $\delta_L(\bar{C}, w) \in F_L$ iff $\delta'(\bar{q}', w) \in F'$, for all $w \in \Sigma^*$. In particular, $\delta_L(\bar{C}, aw') \in$

F_L iff $\delta'(\bar{q}', aw') \in F'$, for all $w' \in \Sigma^*$. Consider $q' = \delta(\bar{q}', a)$. We have, for each string $w \in \Sigma^*$,

$$\begin{aligned}
&\delta_L(C, w) \in F_L \\
&\iff \delta_L(\bar{C}, aw) \in F_L \qquad (\text{since } \delta_L(\bar{C}, a) = C) \\
&\iff \delta'(\bar{q}', aw) \in F' \qquad (\text{induction hypothesis}) \\
&\iff \delta'(q', w) \in F' \qquad (\text{since } \delta'(\bar{q}', a) = q')
\end{aligned}$$

This proves the claim.

Now, choose for each state C of D_L, with the help of the claim, a state q'_C of D' such that $\delta_L(C, w) \in F_L$ iff $\delta'(q'_C, w) \in F'$, for all $w \in \Sigma^*$. Then it holds that $q'_{C_1} \neq q'_{C_2}$ if $C_1 \neq C_2$, for all $C_1, C_2 \in Q_L$. For, suppose $q'_{C_1} = q'_{C_2}$ for some $C_1, C_2 \in Q_L$. Then we have $\delta_L(C_1, w) \in F_L$ iff $\delta_L(C_2, w) \in F_L$, for all $w \in \Sigma^*$. Now pick, $q_1 \in C_1$, $q_2 \in C_2$. Then, by Lemma 2.47, we have $\delta(q_1, w) \in F$ iff $\delta_L(C_1, w) \in F_L$, and $\delta(q_2, w) \in F$ iff $\delta_L(C_2, w) \in F_L$, for all $w \in \Sigma^*$. Thus $\delta(q_1, w) \in F$ iff $\delta(q_2, w) \in F$, for all $w \in \Sigma^*$. Hence, $q_1 \approx_L q_2$ and $C_1 = C_2$, since C_1 and C_2 as states of Q_L are equivalence classes of $\approx_L$.

We conclude that all states $q'_C \in Q'$, for $C \in Q_L$, are different, i.e. the mapping $C \mapsto q'_C$ from Q_L to Q' is an injection. Hence D' has at least as many states as D_L. $\square$

To smoothen the proof, the theorem above assumes that the DFA D started from has reachable states only. This is by no means essential. Given any DFA D having or not having non-reachable states, we can first restrict to its reachable states, and next take the quotient automaton. This leads to a minimal representation of D, i.e. to a DFA D_{min} that accepts the same language and is minimal in the number of states.

Corollary 2.49. Let D be a DFA with accepted language L, let D° be its restriction to reachable states, and D°_L be the quotient DFA of D°. Then there exists no DFA that accepts L and has fewer states than D°_L.

Proof. Since $\mathcal{L}(D^\circ) = L$, the quotient DFA D°_L is well-defined. According to Theorem 2.48, every DFA accepting L has as least as many states as D°_L. $\square$

With the above results in available, we can construct a minimal DFA representation for a regular language L given by a DFA accepting it, if we can find the L-equivalence classes of the set of states. Rather than checking each pair of states against the Definition 2.43 which involves strings of arbitrary length, we can identify the equivalence classes by stepwise refinement. We start from two so-called blocks, one block holding all non-final reachable states and another block holding all final states, and split blocks into smaller subblocks by checking single transitions, thus involving one-letter words only. Before we describe the general algorithm, we first discuss an example.

Example 2.50. Consider the DFA of Figure 2.22, accepting the set of strings over $\{a, b\}$ containing a substring baa. The DFA has set of states $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$. Initially, we consider two blocks of states, viz. $B_{012456} = \{q_0, q_1, q_2, q_4, q_5, q_6\}$ and $B_{37} = \{q_3, q_7\}$, the one containing all non-final reachable states, the other containing all final

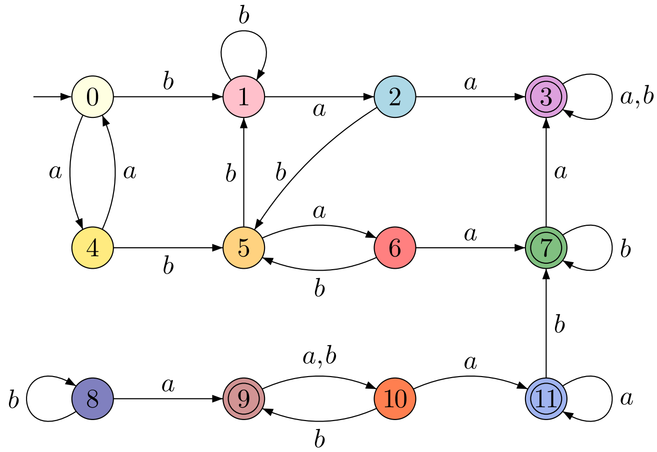


Figure 2.22: A DFA accepting $(a+b)^*{\cdot}baa{\cdot}(a+b)^*$

reachable states. Note, the non-reachable states q_8, q_9, q_{10} and q_{11} are not considered. Next we determine for each state in the blocks to what blocks, rather than states, an a-transition and a b-transition is possible. E.g., state q_0 has an a-transition to state q_4 of block B_{012456}, and a b-transition to state q_1 also of block B_{012456}. State q_2 however, has an a-transition to state q_3 of block B_{37}, and a b-transition to state q_5 of block B_{012456}. This leads to the following table, with two columns for the blocks B_{012456} and B_{37}, eight rows for each individual reachable state.

	012456	37
0	a,b	
1	a,b	
2	b	a
4	a,b	
5	a,b	
6	b	a
3		a,b
7		a,b

We conclude that states q_0, q_1, q_4 and q_5 cannot be L-equivalent to states q_3 and q_7, $\delta(q,a) \in B_{012456} \subseteq Q \backslash F$ for $q = q_0, q_1, q_4, q_5$, while $\delta(q,a) \in B_{37} \subseteq F$ for $q = q_3, q_7$. Therefore, we split the block B_{012456} into two parts, viz. block $B_{0145} = \{q_0,\, q_1,\, q_4,\, q_5\}$ and block $B_{26} = \{q_3,\, q_7\}$. Then, we determine again for each state to what blocks an a-transition and a b-transition is possible. We get the following table, now with three columns corresponding to the three blocks B_{0145}, B_{26} and B_{37}.

	0145	26	37
0	a,b		
1	b	a	
4	a,b		
5	b	a	
2	b		a
6	b		a
3			a,b
7			a,b

We see that states q_0 and q_4 on the one hand, and states q_1 and q_5 on the other hand, show different rows. We have $\delta(q,a) \in B_{0145} \subseteq Q \backslash F$ for $q = q_0, q_4$, but $\delta(q,a) \in B_{37} \subseteq F$ for $q = q_1, q_5$. Therefore, we split block B_{0145} into $B_{04} = \{q_0, q_4\}$ and $B_{15} = \{q_1, q_5\}$. The other blocks don't need to be split; the rows are identical for each of these blocks. Adding another column, now distinguishing B_{04}, B_{15}, B_{26}, and B_{37}, we obtain the following transition table.

	04	15	26	37
0	a	b		
4	a	b		
1		b	a	
5		b	a	
2		b		a
6		b		a
3				a,b
7				a,b

Now, there is within each block no distinguishing row: we have found the L-equivalence classes of the set of states Q. These are the blocks we have now, viz. $\{q_0, q_4\}$, $\{q_1, q_5\}$, $\{q_2, q_6\}$, and $\{q_3, q_7\}$. Note, in the analysis above we have only considered single symbols to establish the L-equivalence classes, rather than strings of arbitrary length.

Completing the quotient construction, (i) we take the equivalence class $\{q_0, q_4\}$ as initial state C_0, because its contains the original initial state q_0, (ii) we have the equivalence class $\{q_3, q_7\}$ as the only final states, since it is the only equivalence class containing final states, thus $Q_F = \{\{q_3, q_7\}\}$, (iii) we have inherited transitions, e.g. $\delta_L(\{q_0, q_4\}, a) = \{q_0, q_4\}$ since $\delta(q_0,a) = q_4 \in \{q_0, q_4\}$ (as well as $\delta(q_4,a) = q_0 \in \{q_0, q_4\}$) and $\delta_L(\{q_0, q_4\}, b) = \{q_1, q_5\}$ since $\delta(q_0,b) = q_5 \in \{q_1, q_5\}$. The resulting quotient DFA is the smallest DFA in number of states which accepts the regular language $(a+b)^* \cdot baa \cdot (a+b)^*$, and is depicted in Figure 2.23.

Pseudo-code for the general DFA minimization algorithm is given in Figure 2.24. We assume that the given DFA $D = (Q, \Sigma, \delta, q_0, F)$ has reachable states only. Otherwise, a reachability algorithm should be run first.

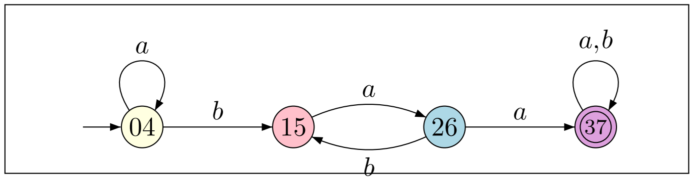


Figure 2.23: Minimal DFA accepting $(a+b)^{*}\cdot baa\cdot(a+b)^{*}$

```
// Q contains reachable states only
// F non-empty
P = { Q\F, F }, continue = true
while continue do
   P' = ∅, continue = false
   for all B in P do
      for all q ∈ B do
         for all a ∈ Σ compute and store δ_P(q, a)
      end for
      split B into non-empty B_1, ..., B_k such that
         ∀q, q' ∈ B ∃i, 1 ⩽ i ⩽ k: q, q' ∈ B_i ⟺ δ_P(q, a) = δ_P(q', a)
      P' = P' ∪ { B_1, ..., B_k }
      if k > 1 then continue = true
   end for
   P = P'
end while
```

Figure 2.24: DFA minimization algorithm

We maintain a partitioning P of the set of states Q into non-empty and pairwise disjoint subsets. Initially, P consists of two blocks, viz. $Q \backslash F$ and F. Note, this requires the set F to be non-empty. However, the minimal equivalent of a DFA without final states is a one-state DFA where the initial state is non-final. We try to refine the partitioning until no more blocks are split. To keep track of this, the progress variable `continue` is maintained, and initially set to `true`.

In the body of the loop, we build a new partitioning P'. We start from the empty set of blocks. For each block B of P we will add one or more blocks $B_1, \ldots, B_k$ to P' covering the same states as B does. We set the progress variable to `false` at the beginning of the body; we only need to go into another iteration if a block was split into two or more subblocks. If a block is non-trivially split, we set with this aim the progress variable `continue` to `true`.

We check for each block B of the current partitioning P whether it should be split or not. To this end we first determine, for each state q in B, to which blocks of P its transitions lead. The function $\delta_P : Q \times \Sigma \to P$ is given by $\delta_P(q, a) = B$ iff $\delta(q, a) \in B$, for $q \in Q$, $a \in \Sigma$, $B \in P$. Since P is a partitioning with non-overlapping blocks, the block $\delta_P(q, a)$ is always well-defined. We store the values $\delta_P(q, a)$ as we need them when splitting the block B.

Next, we group the states of B in subblocks with equal values $\delta(\cdot, a)$, for $a \in \Sigma$. We can do so, by taking a state $q \in B$ and put it aside along with all states $q' \in B$ such that $\delta_P(q, a) = \delta_P(q', a)$, for all $a \in \Sigma$. From the rest of B, if non-empty, we pick a state again, say $q'' \in B$, and put it aside along with all states $q''' \in B$ such that $\delta_P(q''', a) = \delta_P(q'''', a)$, etc. If we are done after k steps, we have our blocks $B_1, \ldots, B_k$. Note, all states of B occur in these blocks, and no block contains other states. Moreover, by construction, the blocks $B_1, \ldots, B_k$ are non-empty and pairwise disjoint. We add the blocks $B_1, \ldots, B_k$ to the new growing partitioning P'. If the split up was non-trivial, i.e. B is split into more than one subblock, we set the progress variable `continue` to `true`, since if $k > 1$, the new blocks $B_1, B_2, \ldots, B_k$, may lead to a split up elsewhere. Then we are done with block B, and continue the `for`-loop of line 6. with the next block, if applicable.

After all blocks have been checked on splitting, we overwrite the partitioning P with the partitioning P'. If P' is strictly finer than P, the progress variable was set to `true` underway, and we iterate the outer `while`-loop. If not, the algorithm terminates. Note that we cannot refine the initial partitioning *ad infinitum*. Finer grained that the partitioning of singletons we cannot go. Likely, the algorithm will stop earlier.

The correctness of the algorithm follows from the following theorem.

Theorem 2.51. Let $D = (Q, \Sigma, \delta, q_0, F)$ be a DFA with reachable states only. Define the relations $\equiv_n \subseteq Q \times Q$, for $n \geqslant 0$, by

$$\begin{aligned} q \equiv_0 q' &\iff q, q' \in Q \backslash F \;\vee\; q, q' \in F \\ q \equiv_{n+1} q' &\iff q \equiv_n q' \;\wedge\; \forall a \in \Sigma\colon \delta(q, a) \equiv_n \delta(q', a) \end{aligned}$$

Then the following statements hold true.

(a) Each relation $\equiv_n$, for $n \geqslant 0$, is an equivalence relation on Q.

(b) For all $q, q' \in Q$: $q \equiv_n q'$ iff $\forall w \in \Sigma^*, |w| \leqslant n \colon \delta(q, w) \in F \iff \delta(q', w) \in F$.

(c) Let P_n be the partitioning P after n iterations of the algorithm. Then for all $q, q' \in Q$: $q \equiv_n q'$ iff $\exists B \in P_n \colon q, q' \in B$.

(d) If $\equiv_{n+1} = \equiv_n$ for some $n \geqslant 0$, then $\equiv_{n+k} = \equiv_n$, for all $k \geqslant 0$.

(e) If $q, q' \in B$ for a block B of the final partitioning P_n of the algorithm, then it holds that $q \approx_L q'$.

Proof. We leave part (a) as an exercise. We prove part (b) by induction on n. Basis, $n = 0$: Clear, since $\delta(q, \varepsilon) = q$, $\delta(q', \varepsilon) = q'$, and $q \equiv_0 q'$ iff both $q, q' \in Q \backslash F$ or $q, q' \in F$. Induction step, $n + 1$: For $q, q' \in Q$ we have

$$
\begin{aligned}
& q \equiv_{n+1} q' \\
\iff\ & q \equiv_n q' \wedge \forall a \in \Sigma \colon \delta(q, a) \equiv_n \delta(q', a) && \text{(by definition of } \equiv_{n+1}\text{)} \\
\iff\ & \forall w \in \Sigma^*, |w| \leqslant n \colon \delta(q, w) \in F \Leftrightarrow \delta(q', w) \in F \wedge \\
& \qquad \forall a \in \Sigma\, \forall w \in \Sigma^*, |w| \leqslant n \colon \delta(\delta(q, a), w) \in F \Leftrightarrow \delta(\delta(q', a), w) \in F \\
& && \text{(by definition of } \equiv_n \text{ twice)} \\
\iff\ & \forall w \in \Sigma^*, |w| \leqslant n \colon \delta(q, w) \in F \Leftrightarrow \delta(q', w) \in F \wedge \\
& \qquad \forall w \in \Sigma^*, 1 \leqslant |w| \leqslant n{+}1 \colon \delta(q, w) \in F \Leftrightarrow \delta(q', w) \in F \\
& && \text{(by definition of } \delta(q, aw) \text{ and } \delta(q', aw)\text{)} \\
\iff\ & \forall w \in \Sigma^*, |w| \leqslant n{+}1 \colon \delta(q, w) \in F \Leftrightarrow \delta(q', w) \in F
\end{aligned}
$$

Part (c) is proven by induction on n. Basis, $n = 0$: Clear by definition of $\equiv_0$ and initialization of P. Induction step, $n + 1$: For $q, q' \in Q$ we have

$$
\begin{aligned}
& q \equiv_{n+1} q' \\
\iff\ & q \equiv_n q' \wedge \forall a \in \Sigma \colon \delta(q, a) \equiv_n \delta(q', a) && \text{(definition } \equiv_{n+1}\text{)} \\
\iff\ & \exists B_n \in P_n \colon q, q' \in B_n \wedge \forall a \in \Sigma\, \exists B_a \in P_n \colon \delta(q, a), \delta(q', a) \in B_a \\
& && \text{(induction hypothesis twice)} \\
\iff\ & \exists B_n \in P_n \colon q, q' \in B_n \wedge \forall a \in \Sigma \colon \delta_{P_n}(q, a) = \delta_{P_n}(q', a) \\
& && \text{(definition } \delta_{P_n}\text{)} \\
\iff\ & \exists B_{n+1} \in P_{n+1} \colon q, q' \in B_{n+1} && \text{(definition of the algorithm)}
\end{aligned}
$$

Part (d) is shown by induction on k. Assume $\equiv_{n+1} = \equiv_n$. Basis, $k = 0$: Clear. Induction step, $k + 1$: We have, for $q, q' \in Q$,

$$
\begin{aligned}
& q \equiv_{n+k+1} q' \\
\iff\ & q \equiv_{n+k} q' \wedge \forall a \in \Sigma \colon \delta(q, a) \equiv_{n+k} \delta(q', a) && \text{(definition } \equiv_{n+k+1}\text{)} \\
\iff\ & q \equiv_n q' \wedge \forall a \in \Sigma \colon \delta(q, a) \equiv_n \delta(q', a) && \text{(induction hypothesis)} \\
\iff\ & q \equiv_{n+1} q' && \text{(definition } \equiv_{n+1}\text{)} \\
\iff\ & q \equiv_n q' && \text{(by assumption)}
\end{aligned}
$$

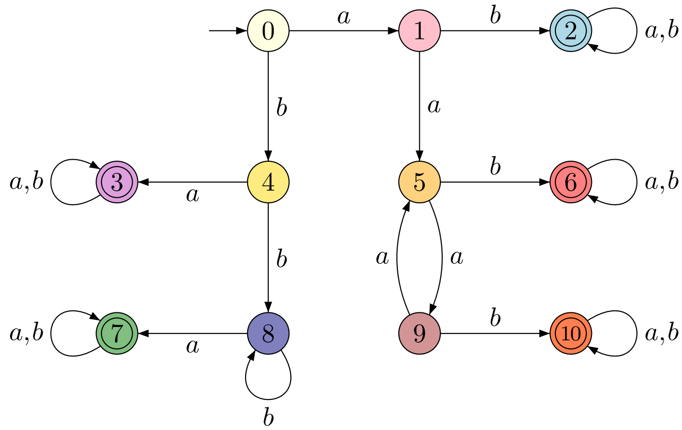


Figure 2.25: A DFA accepting $(a+b)^* \cdot (ab+ba) \cdot (a+b)^*$

For the proof of part (e) we reason as follows: Pick $q, q' \in B$ for some block B of the final partitioning P_n. For P_n it holds that P_{n-1}, since only if all blocks remain unaltered, the progress variable `continue` is not set to `true`. Thus, by part (c), $\equiv_n \; = \; \equiv_{n-1}$. Hence, by part (d), $\equiv_{n-1+k} \; = \; \equiv_{n-1}$, for $k \geqslant 0$, and hence $\equiv_{n+k} \; = \; \equiv_n$, for $k \geqslant 0$. By part (b) we obtain

$$q \equiv_n q' \quad \text{iff} \quad \forall w \in \Sigma^* \colon \delta(q,a) \in F \iff \delta(q',a) \in F$$

Since, by assumption $q, q' \in B$ for some block B of P_n, we have $q \equiv_n q'$, by part (c). It follows that $q \approx_L q'$, by definition of $\approx_L$. □

Example 2.52. As another example of our minimization technique, consider the DFA depicted in Figure 2.25 having $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}\}$ as its set of states. Note, all states are reachable. Again we start with two blocks, the non-final states $B_{014589} = \{q_0, q_1, q_4, q_5, q_8, q_9\}$ and the final states $B_{236710} = \{q_2, q_3, q_6, q_7, q_{10}\}$. Next we determine for each individual state, q_0 to q_{10}, to which blocks their transitions for a and b lead. This is recorded in the left part of the table below.

	014589	236710		0	159	48	236710
0	a,b		0		a	b	
1	a	b	1		a		b
4	b	a	5		a		b
5	a	b	9		a		b
8	b	a	4			b	a
9	a	b	8			b	a
2		a,b	2				a,b
3		a,b	3				a,b
6		a,b	6				a,b
7		a,b	7				a,b
10		a,b	10				a,b

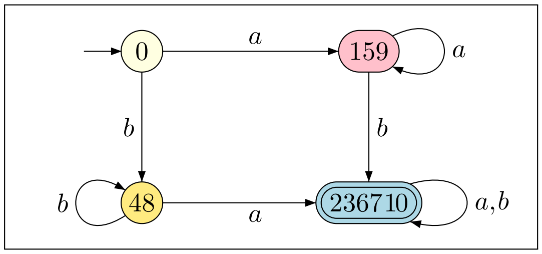


Figure 2.26: Minimal DFA accepting $(a+b)^{*} \cdot (ab+ba) \cdot (a+b)^{*}$

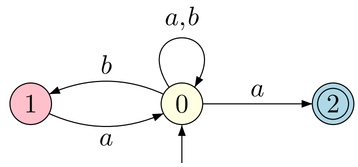


Figure 2.27: An NFA for which quotienting does not work

We see that block B_{014589} splits in three subblocks, B_0, B_{159} and B_{48}, while the block B_{236710} remains as is. The next iteration does not lead to further refinement. The resulting quotient DFA, with a minimal number of states, is given in Figure 2.26. The insight is that a word with first symbol a will be accepted if a symbol b follows at some point, and likewise, a word starting with b will be accepted if an a occurs after zero or more b's.

Concluding the section we show that the quotient construction of dividing out by L-equivalence does not work for minimization of NFA. Consider the NFA $\mathcal{N}$ of Figure 2.27 accepting the language $L = (a+b)^{*} \cdot a$. We adapt Definition 2.43 for $\mathcal{N} = (Q, \Sigma, \delta, q_0, F)$ as follows: $q \approx_L q'$ iff $\forall w \in \Sigma^{*}$: $\delta(q,w) \cap F = \emptyset \iff \delta(q',w) \cap F = \emptyset$. Using this definition we see that the states q_0 and q_1 are not L-equivalent: $\delta(q_0, a) = \{q_0, q_2\}$ thus $\delta(q_0, a) \cap F \neq \emptyset$, but $\delta(q_1, a) = \{q_0\}$ thus $\delta(q_1, a) \cap F = \emptyset$. Also, both q_0 and q_1 are not L-equivalent to q_2: $\delta(q_0, \varepsilon), \delta(q_1, \varepsilon) \cap F = \emptyset$, but $\delta(q_2, \varepsilon) \cap F \neq \emptyset$. Thus, all three states q_0, q_1 and q_2 are pairwise not L-equivalent. However, $\mathcal{N}$ is not minimal in the number of states. The NFA obtained from $\mathcal{N}$ by deleting the state q_1 consists of two states, i.e. one less, and also accepts L.

2.5.1 Exercises for Section 2.5

Exercise 2.5.28. Let D be a DFA accepting the language L. Prove that the relation $\approx_L \subseteq Q \times Q$ of L-equivalence for D is an equivalence relation.

Answer to Exercise 2.5.28 Let $D = (Q, \Sigma, \delta, q_0, F)$ be a DFA accepting the language L. We check that $\approx_L$ is an equivalence relation, i.e. that $\approx_L$ is reflexive, symmetric and transitive.

Reflexivity: For $q \in Q$, clearly $\delta(q,w) \in F$ iff $\delta(q,w)$, for all $w \in \Sigma^{*}$. Thus $q \approx_L q$, for all $q \in Q$.

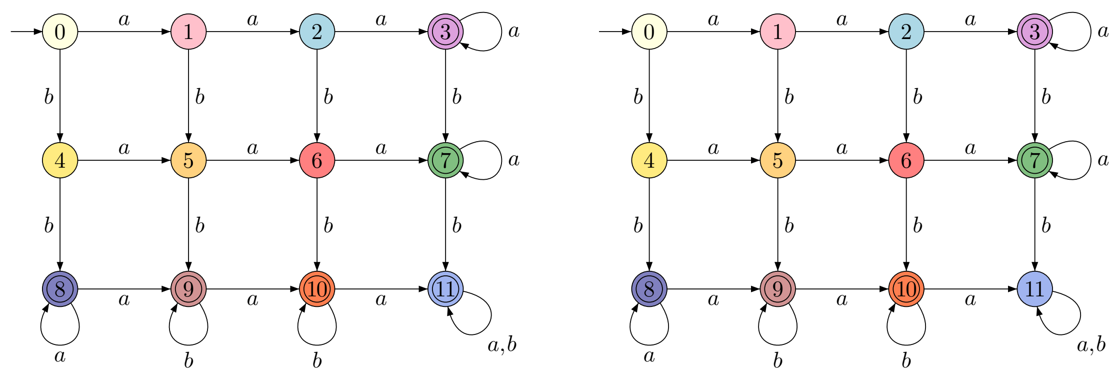


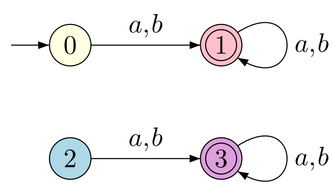
Figure 2.28: Two DFA for Exercise 2.5.30

Symmetry: Suppose $q_1, q_2 \in Q$ such that $q_1 \approx_L q_2$. By definition of $\approx_L$ we have $\delta(q_1, w) \in F$ iff $\delta(q_2, w) \in F$, for all $w \in \Sigma^*$. Thus, $\delta(q_2, w) \in F$ iff $\delta(q_1, w) \in F$, for all $w \in \Sigma^*$. Hence, $q_2 \approx_L q_1$.

Transitivity: Suppose $q_1 \approx_L q_2$ and $q_2 \approx_L q_3$ for $q_1, q_2, q_3 \in Q$. For arbitrary $w \in \Sigma^*$ we have $\delta(q_1, w) \in F$ iff $\delta(q_2, w) \in F$, and $\delta(q_2, w) \in F$ iff $\delta(q_3, w) \in F$. Thus $\delta(q_1, w) \in F$ iff $\delta(q_3, w) \in F$, for all $w \in \Sigma^*$. Hence $q_1 \approx_L q_3$.

Exercise 2.5.29. Give an example of a DFA accepting a language L with four states in total, two reachable and two non-reachable, where each reachable state is L-equivalent to a non-reachable state.

Answer to Exercise 2.5.29



Exercise 2.5.30. Consider the two DFA of Figure 2.28 with accepted languages L_ℓ and L_r. In the left automaton state q_{11} is accepting, in the right automaton it is not.

(a) For the automaton on the left, how many states are L_ℓ-equivalent to state q_0, to state q_5, to state q_{10}, and to state q_{11}, respectively.

(b) For the automaton on the right, how many states are L_r-equivalent to state q_0, to state q_3, to state q_9, and to state q_{11}, respectively.

Answer to Exercise 2.5.30

(a) In the DFA on the left of Figure 2.28, state q_0 is L_ℓ-equivalent to itself only. The same for state q_5. All final states are L_ℓ-equivalent to state q_{10} and state q_{11}.

(b) In the DFA on the right of Figure 2.28 each state is only L_r-equivalent to itself.

Exercise 2.5.31. Let D be a DFA accepting the language L. Obtain the DFA D° by deleting the non-reachable states from D. Clearly, D° accepts L too. Consider the quotients D_L and D°_L, respectively.

(a) Suppose $q \in C$ for a state q of D and a state C of D_L. If the string $w \in \Sigma^*$ is such that $\delta(q_0, w) = q$, then $\delta_L(C_0, w) = C$.

(b) Prove that a non-reachable state of D_L, if present, consists of non-reachable states of D only.

(c) Prove that each state of D°_L is reachable.

(d) Conclude that is doesn't make an essential difference for the construction of a minimal DFA if the non-reachable states are removed before or after the quotient construction.

Answer to Exercise 2.5.31

(a) Induction on the length of w. Basis, $|w| = 0$: Then $w = \varepsilon$. Thus $q = q_0$, and $C = C_0$ since $q_0 = q \in C$. Induction step, $|w| > 0$: Pick $a \in \Sigma$ and $v \in \Sigma^*$ such that $w = va$. Put $\bar{q} = \delta(q_0, v)$. Then $q = \delta(\bar{q}, a)$. Suppose $\bar{q} \in \bar{C}$. By induction hypothesis we have $\delta(C_0, v) = \bar{C}$. Since $\bar{q} \in \bar{C}$, $q \in C$, and $q = \delta(\bar{q}, a)$ it follows that $C = \delta_L(\bar{C}, a)$. Therefore, $\delta_L(C_0, w) = \delta_L(C_0, va) = \delta_L(\bar{C}, a) = C$.

(b) By part (a), if $q \in Q$ is reachable in D and $q \in C$, then $C \in Q_L$ is reachable in D_L. Put differently, if $C \in Q_L$ is not reachable in D_L, then C contains no reachable state of D.

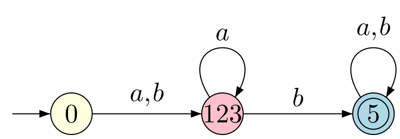
(c) By construction of D°_L each state $C \in Q_L$ is non-empty and contains reachable states of D° only. Thus, by part (a), each state $C \in Q_L$ is reachable in D°_L.

(d) On the one hand, D°_L is the smallest DFA accepting L. On the other hand, by parts (b) and (c), D°_L can be seen as obtained from D_L by removing non-reachable states.

Exercise 2.5.32. Construct a DFA with three states that is language equivalent to the DFA given in Figure 2.29.

Answer to Exercise 2.5.32

	0123	4		0	123	4
0	a,b		1		a	b
1	a	b	2		a	b
2	a	b	3		a	b
3	a	b				



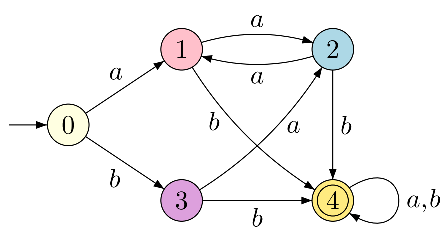


Figure 2.29: DFA for Exercise 2.5.32

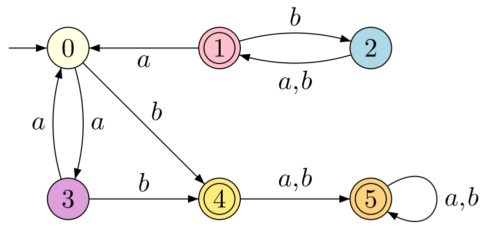


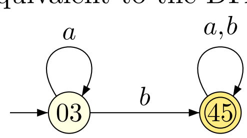
Figure 2.30: DFA for Exercise 2.5.33

Exercise 2.5.33. Construct a DFA with two states that is language equivalent to the DFA given in Figure 2.30.

Answer to Exercise 2.5.33 First restrict to reachable states only, i.e. to q_0, q_3, q_4, and q_5, and discarding q_1 and q_2. Next, compute L-equivalence classes.

	03	45
0	a	b
3	a	b
4		a,b
5		a,b

No further split of the initial blocks $\{q_0, q_3\}$ and $\{q_4, q_5\}$. This leads to the following minimal DFA that is language equivalent to the DFA of Figure 2.30.



Exercise 2.5.34. Construct a DFA with a minimal number of states that is language equivalent to the DFA given in Figure 2.31.

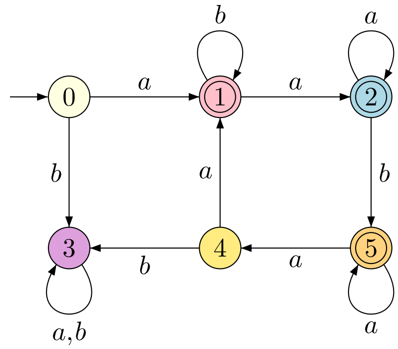
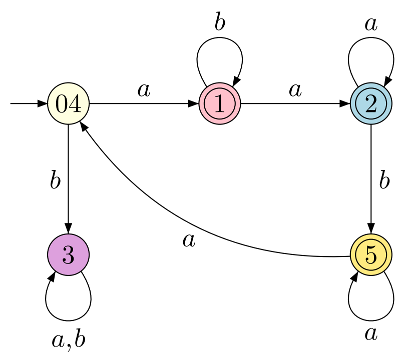


Figure 2.31: DFA for Exercise 2.5.34

Answer to Exercise 2.5.34

	034	125		04	12	3	5	04	1	2	3	5
0	b	a	0		a	b	0		a		b	
3	a,b		4		a	b	4		a		b	
4	b	a	1		a,b							
1		a,b	2		a		b					
2		a,b										
5	b	a										



Exercise 2.5.35. Construct a DFA with a minimal number of states that is language equivalent to the DFA given in Figure 2.32.

Answer to Exercise 2.5.35

	0234	1		03	24	1		03	2	4	1
0	a	b	0		a	b	0		a		b
2	a,b		3		a	b	3			a	b
3	a	b	2	a,b							
4	a,b		4	a	b						

The DFA of Figure 2.32 is minimal already.

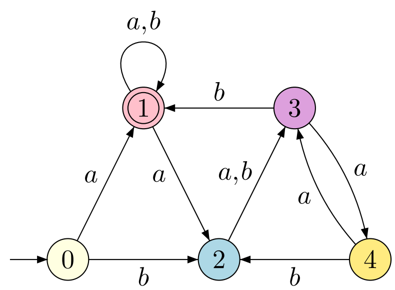


Figure 2.32: DFA for Exercise 2.5.35