Solutions to Supplementary Problems

Problem S4.1 Simplify the following regular expressions, i.e. design simpler expressions describing the same languages:

1. $(\emptyset^* \cup a)(a^*)^*(b \cup a)b^*$
2. $(a \cup b)^* \cup \emptyset \cup (a \cup b)b^*a^*$
3. $a(b^* \cup a^*)(a^*b^*)^*$

Solution.

1. $$\begin{aligned}(\emptyset^* \cup a)(a^*)^*(b \cup a)b^* &= (\varepsilon \cup a)(a^*)^*(b \cup a)b^* \\ &= (\varepsilon \cup a)a^*(b \cup a)b^* && \mathcal{L}((a^*)^*) = \mathcal{L}(a^*) \\ &= a^*(b \cup a)b^* && \mathcal{L}((\varepsilon \cup a)a^*) = \mathcal{L}(a^*) \\ &= a^*(a \cup b)b^*\end{aligned}$$

2. $$(a \cup b)^* \cup \emptyset \cup (a \cup b)b^*a^* = (a \cup b)^*$$

   The result can be immediately deduced from the fact that the first subexpression of the union already generates all words in Σ^*.

3. $$a(b^* \cup a^*)(a^*b^*)^* = a(a \cup b)^*$$

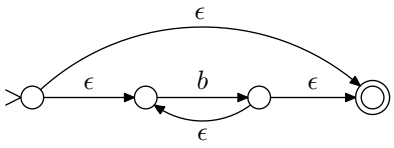
   Here we note that since subexpression $r_2 = (a^*b^*)^*$ covers all strings that are covered by $r_1 = (b^* \cup a^*)$ and $\varepsilon \in \mathcal{L}(r_1)$, it is the case that $r_2 \subseteq r_1r_2 \subseteq r_2r_2 = r_2$, and hence $r_1r_2 = r_2 = (a \cup b)^*$.

Problem S4.2 Determine whether the regular expressions $r_1 = b^*a(a^*b^*)^*$ and $r_2 = (a \cup b)^*a(a \cup b)^*$ are equivalent (i.e. describe the same language), by constructing the minimal deterministic finite automata corresponding to them.

Solution. For each regular language there exists a unique[1] minimal deterministic finite automaton. Thus, we can check the equivalence of two regular expressions by generating their corresponding minimal automata, and then checking whether they are identical.

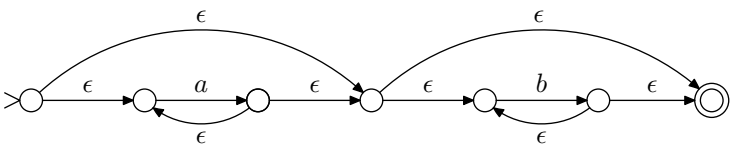
We first construct the automaton corresponding to the regular expression r_1:

b^*:

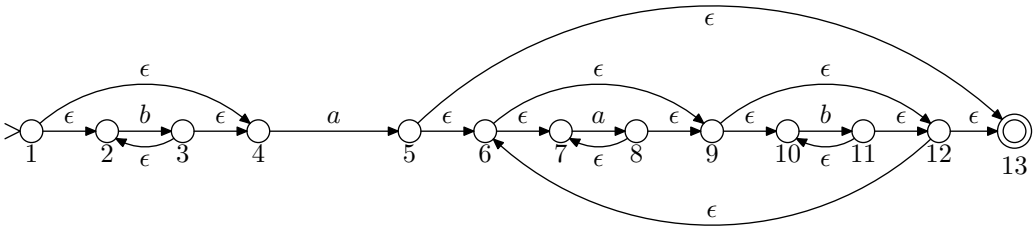


[1] Up to the naming of the states.

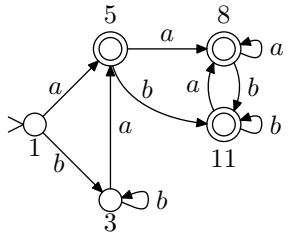
a^*b^***:**



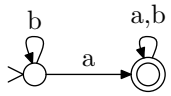
$r_1 = b^*a(a^*b^*)^*$**:**



Removing the ε-transitions from this automaton yields:

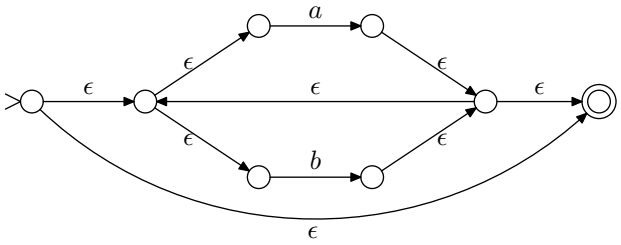


Since this automaton is already deterministic, we may skip determinisation and move directly to the minimisation phase. The minimisation algorithm merges states 1 and 3 together, as well as states 5, 8 and 11. The resulting minimal automaton M_{r_1} is thus:

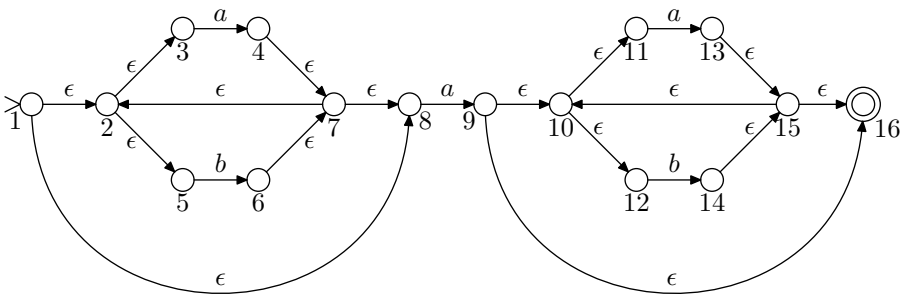


Next we do the same construction for r_2:

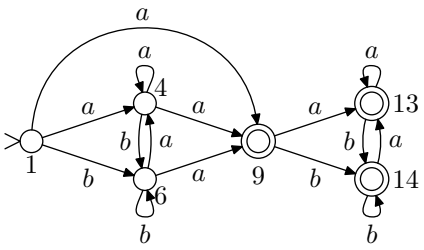
$(a \cup b)^*$**:**



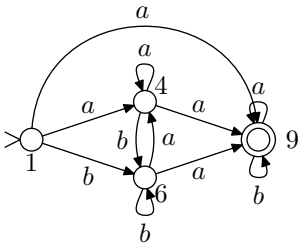
$r_2 = (a \cup b)^* a (a \cup b)^*$**:**



By removing the ε-transitions we get:

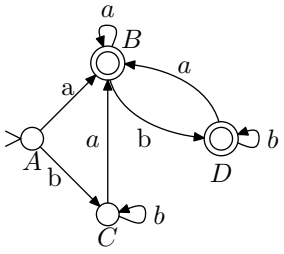


We note that this automaton may be simplified by combining all accepting states into one:

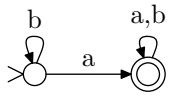


Next, we determinise this automaton:

Nondet. state set	a	b	New name
$\{1\}$	$\{4,9\}$	$\{6\}$	$\to A$
$\{4,9\}$	$\{4,9\}$	$\{6,9\}$	$B \to$
$\{6\}$	$\{4,9\}$	$\{6\}$	C
$\{6,9\}$	$\{4,9\}$	$\{6,9\}$	$D \to$



When we minimise this automaton, the result is again:



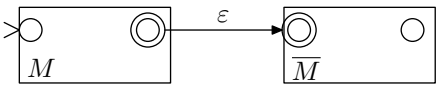
Because both r_1 and r_2 lead to the same minimal automaton, the languages $\mathcal{L}(r_1)$ and $\mathcal{L}(r_2)$ are the same, and the expressions are thus equivalent.

Problem S4.3 Prove that if L is a regular language, then so is $L' = \{xy \mid x \in L, y \notin L\}$.

Solution. The easiest way to prove that a language is regular is usually to demonstrate a finite automaton that recognises it. In the present problem we have some arbitrary regular language L that is used to define a new language:

$$L' = \{xy \mid x \in L \text{ and } y \notin L\}.$$

Because L is regular, there must be some deterministic finite automaton M that recognises it. Let $\overline{M}$ be an automaton for the complement of the language L. Such an $\overline{M}$ can be formed by flipping the accepting status of each state in M. This makes $\overline{M}$ accept exactly the words that M rejects. Let us now compose automata M and $\overline{M}$ into an automaton M' by adding an ε-transition from each accepting state of M to the initial state of $\overline{M}$. This M' now recognises the language L', and thus witnesses that L' is regular.



Proofs like this can also be done by applying closure properties of languages. Regular languages are closed under union, catenation, Kleene star, complementation and intersection. Since L' is a catenation of L and the complement of L ($L' = L\overline{L}$), it is regular because of the closure properties above.

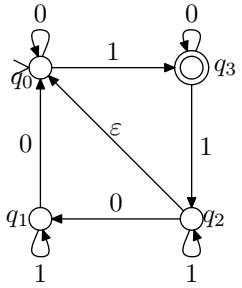
Note, however, that the properties only guarantee closure in one direction. For instance, if A is a regular language that can be represented as the union of B and C, nothing can be deduced about the languages B and C. Consider the following example:

$$B = \{a^n b^n \mid n \geq 0\}$$
$$A = B \cup \overline{B} = \Sigma^*$$

Now the language A is regular, but B and $\overline{B}$ are both nonregular. The same caution should observed when applying closure under catenation, Kleene star, and intersection: regularity of the components implies the regularity of the composite, but not the other way round.

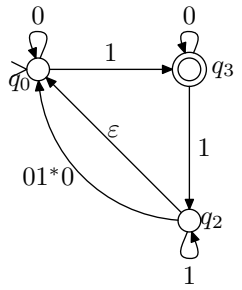
Appendix. Constructing a regular expression from a finite automaton.

As an additional example we construct a regular expression that corresponds to the ε-automaton:



We start by removing all states from which no accepting state is reachable. In this case there aren't any. If the automaton has more than one accepting state, they are made non-accepting and a new accepting state is added with an ε-transition from each previously accepting state. As the automaton in this example has only one accepting state, this step is not needed either.

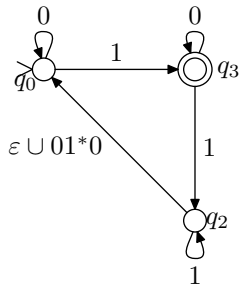
The next step is to reduce the automaton by removing states one at a time, until only the initial state and the unique accepting state are left. (These may in some cases also be the same state.) Let's start with state q_1:



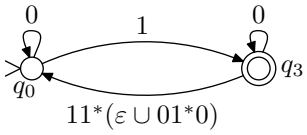
Transitions that are unrelated to q_1 remain as they are. Every transition path that goes through q_1 is replaced by a direct transition on an expression that is obtained as the catenation of the expressions on the path. In this case $q_2 \xrightarrow{01^*0} q_0$ comes from combining $q_2 \xrightarrow{0} q_1$, $q_1 \xrightarrow{1} q_1$ and $q_1 \xrightarrow{0} q_0$. Intuitively this means that q_0 is reachable from q_2 by first reading one 0, after which an arbitrary number of 1's can be read by looping in q_1, and finally one 0 is read. There are no other paths through q_1.

From state q_2 there are now two alternative transitions to q_0. These can be

combined as follows:



Next we remove state q_2 similarly as earlier:



Now the desired regular expression can be read from the transitions. The basic idea is to check every possible path from the initial state to the accepting state. The expression starts with a transition from the initial state to the accepting state, and ends with all possible ways to loop in the accepting state:

$$\underbrace{0^*1}_{q_0 \to^* q_1}\underbrace{(0 \cup 11^*(\varepsilon \cup 01^*0)0^*1)^*}_{q_1 \to^* q_1}$$