1 Finite Automata and Regular Languages [Exam]

a) We could use the systematic transformation scheme presented in the lecture (slide 1/75). Considering the large number of states, however, this will easily lead to an explosion of states in the derandomized automaton. Hence, we build the deterministic finite automaton in a step-wise manner, only creating those states that are actually required: Initially, the automaton requires a 0. Subsequently, only a 1 is accepted. Including the various transitions, this 1 can lead to three different states, namely states 2, 3, and 4.

\[
\begin{align*}
(1) & \xrightarrow{0} (2,4) \xrightarrow{1} (2,3,4)
\end{align*}
\]

In any of the states 2, 3, and 4, only a 1 is accepted. Assume that the automaton is currently in state 2, this 1 can lead to states \{2,3,4\} when including all \(\varepsilon\)-transitions. When in state 3, the 1 leads to states \{2,3,4,5\} and finally, when being in state 4, the reachable states given a 1 are \{2,3,4\}. Hence, a 1 leads from state \{2,3,4\} to state \{2,3,4,5\}. Repeating the same process for state \{2,3,4,5\}, we can see that, again, only a 1 is accepted, which leads to state \{2,3,4,5,6\}. Because the state 6 in the original NFA was an accepting state, \{2,3,4,5,6\} is also accepting in the DFA. From state \{2,3,4,5,6\}, an additional 1 will lead to another accepting state \{1,2,3,4,5,6\}. And from this state, any subsequent 1 returns to state \{1,2,3,4,5,6\} as well.

\[
\begin{align*}
(1) & \xrightarrow{0} (2,4) \xrightarrow{1} (2,3,4) \xrightarrow{1} (2,3,4,5) \xrightarrow{1} (2,3,4,5,6) \xrightarrow{1} (1,2,3,4,5,6)
\end{align*}
\]

What happens if a 0 occurs in the input? This is feasible only when the deterministic state includes either state 1 or state 6. In state \{2,3,4,5,6\}, a 0 necessarily leads to state \{4\}, whereas in state \{1,2,3,4,5,6\} a 0 leads to state \{2,4\}. In both of these states, the only acceptable input symbol is a 1 and leads to the state \{2,3,4\}. Hence, the deterministic finite automaton looks like this:

\[
\begin{align*}
(1) & \xrightarrow{0} (2,4) \xrightarrow{1} (2,3,4) \xrightarrow{1} (2,3,4,5) \xrightarrow{1} (2,3,4,5,6) \xrightarrow{1} (1,2,3,4,5,6)
\end{align*}
\]
It can easily be seen, that first the states \( \{4\}, \{2,4\} \) and then the states \( \{2,3,4,5,6\}, \{1,2,3,4,5,6\} \) can be merged and hence, the automaton can be reduced to the one shown in the next figure.

This is not a DFA yet, because the crash state is still missing. The final deterministic automaton looks like this:

b) By studying the above automaton, it can be seen that the following regular language is accepted: \( 01111^*(01111^*)^* = (01111^+)^+ \).

2 Pumping Lemma [Exam]

The Pumping Lemma in a Nutshell

Given a language \( L \), assume for contradiction that \( L \) is regular and has the pumping length \( p \). Construct a suitable word \( w \in L \) with \( |w| \geq p \) ("there exists \( w \in L \)) and show that for all divisions of \( w \) into three parts, \( w = xyz \), with \( |x| \geq 0, |y| \geq 1 \), and \( |xy| \leq p \), there exists a pumping exponent \( i \geq 0 \) such that \( w' = xy^iz \not\in L \). If this is the case, \( L \) is not regular.

a) Language \( L_1 \) can be shown to be non-regular using the pumping lemma. Assume for contradiction that \( L_1 \) is regular and let \( p \) be the corresponding pumping length. Choose \( w \) to be the word \( 0110^p1^p \). Because \( w \) is an element of \( L_1 \) and has length more than \( p \), the pumping lemma guarantees that \( w \) can be split into three parts, \( w = xyz \), where \( |xy| \leq p \) and for any \( i \geq 0 \), we have \( xy^iz \in L_1 \). In order to obtain the contradiction, we must prove that for every possible partition into three parts \( w = xyz \) where \( |xy| \leq p \), the word \( w \) cannot be pumped. We therefore consider the various cases.
(1) If \( y \) starts with any suffix of the first three symbols (i.e. 011) of \( w \), the word \( w \) cannot be pumped without violating either the constraints \( a = 1 \) or \( b = 2 \) (e.g. 0101101\( \ast \) for \( y = 01) \) or creating a word with an illegal structure (e.g. 01101101\( \ast \) for \( y = 011)).

(2) If \( y \) consists of only 0s from the second block, the word \( w' = xyyz \) has more 0s than 1s in the last \(|w'| - 3 \) symbols and hence \( c \neq d \).

Note that \( y \) cannot contain 1s from the second block because of the requirement \(|xy| \leq p\).

We have shown that for all possible divisions of \( w \) into three parts, the pumped word is not in \( L_1 \). Therefore, \( L_1 \) cannot be regular and we have a contradiction.

b) With the adapted language \( L_2 \), the proof of non-regularity is much more tricky! Specifically, non-regularity of \( L_2 \) cannot be proven using the pumping lemma, because any word in \( L_2 \) can actually be pumped! Consider for instance a word \( w \) of the form 0110\( \ast \). In this case, we can split \( w \) into the three parts \( x = 0, y = 11, z = 0^p1^p \), which is in accordance with the rules of the pumping lemma. It can be seen, however, that any word \( xy'z \) is also in \( L_2 \! \). That is, the language \( L_2 \) can be pumped and yet, it is not regular as shown below.

Assume for contradiction that there exists a finite automaton \( A \) which accepts the language \( L_2 \). Every word that starts with the input-sequence 0110 is only accepted if the remainder of the word has the form 0\( c-1 \)\( \ast \) for some integer \( c > 0 \). Let \( q_1 \) be the state reached after the input 0110. Given the automaton \( A \), we can construct a regular automaton \( A' \) that is equivalent to \( A \) with the only difference that its initial state is \( q_1 \). By the definition of \( A \), this adapted finite automaton \( A' \) accepts all words of the form 0\( c-1 \)\( d^c \). However, as shown on slide 1/95 of the script, the language 0\( c-1 \)\( d^c \) is not regular. Hence, \( A' \) and thus \( A \) cannot be finite automata. Because there exists a finite automaton for every regular language, it follows that \( L_2 \) cannot be regular. Language \( L_2 \) shows that while every regular language can be pumped according to the pumping lemma, there are also non-regular languages that can be pumped.

Variant: We can alternatively use the fact that if two languages \( L \) and \( L' \) are regular, the language defined by the intersection of the two languages \( L \cap L' \) is regular as well (cf. p. 1/41). Consider the regular language \( L_3 = \{ w \in 0110^p \ast \} \). Notice that the intersection of \( L_3 \) with \( L_2 = \{ 0^a1^b0^c1^d \mid a, b, c, d \geq 0 \text{ and if } a = 1 \text{ and } b = 2 \text{ then } c = d \} \) contains exactly all words \( w \in \{ 0110^\ast \} \). This, however, is the exact language \( L_1 \) we proved not to be regular in the first part of this exercise. If we assume \( L_2 \) to be regular, \( L_1 \) must be regular as well, since \( L_1 = L_2 \cap L_3 \). This is a contradiction. Thus \( L_2 \) cannot be regular.

Be Careful!
The argumentation above is based on the closure properties of regular languages and only works in the direction presented. That is, for an operator \( \circ \in \{ \cup, \cap, \ast \} \), we have:

If \( L_1 \) and \( L_2 \) are regular, then \( L = L_1 \cup L_2 \) is also regular.

If either \( L_1 \) or \( L_2 \) or both are non-regular, we cannot deduce the non-regularity of \( L \) or vice-versa. Moreover, \( L \) being regular does not imply that \( L_1 \) and \( L_2 \) are regular as well. This may sound counter-intuitive which is why we give examples for the three operators.

- \( L = L_1 \cup L_2 \): Let \( L_1 \) be any non-regular language and \( L_2 \) its complement. Then \( L = \Sigma^\ast \) is regular.

- \( L = L_1 \cap L_2 \): Let \( L_1 \) be any non-regular language and \( L_2 \) its complement. Then \( L = \emptyset \) is regular.

- \( L = L_1 \ast L_2 \): Let \( L_1 = \{ a^\ast \} \) (a regular language) and \( L_2 = \{ a^p \mid p \text{ is prime} \} \) (a non-regular language) then \( L = \{ aaa^\ast \} \) is regular.

Hence, to prove that a language \( L_2 \) is non-regular, you assume it to be regular for contradiction. Then you combine it with a regular language \( L_r \) to obtain a language \( L = L_2 \circ L_r \). If \( L \) is non-regular, \( L_2 \) could not have been regular either.
a) The regular expression can be obtained from the finite automaton using the transformation presented in the script on slide 1/85. After ripping out state $q_2$, the corresponding GNFA looks like this:

After also removing state $q_1$, the GNFA looks as follows.

Eliminating the last state $q_3$ yields the final solution, which is $(01^*0)^*1(0 \cup 11^*0)(01^*0)^*1^*$.

Note: Ripping out the interior states in a different order yields a distinct yet equivalent regular expression. The order $q_3, q_2, q_1$, for example, results in $((0 \cup 10^*1)^*0)^*10^*$.

b) The best way to solve this problem is to ask, which words are actually not in $\Phi(L)$. The word 1, for instance must be in $\Phi(L)$, because the word 10 is in $L$. Moreover, the word 11 is in $\Phi(L)$, because 1101 is in $L$. Also, 10, 01, and 00 are in $\Phi(L)$ because of the words 1000, 0101, and 0010, respectively. More generally, it can be seen from every state in the automaton and for all $k \geq 2$, there is a sequence of $k$ symbols that lead to the accepting state. Hence, all words of length at least 2 are in $\Phi(L)$. Also, as seen before, the word 1 is in $\Phi(L)$. The only words that are not in $\Phi(L)$ are therefore 0 and $\varepsilon$: 0 is not in $\Phi(L)$, because there is no word of length 2 in $L$ starting with 0 that leads to an accepting state, and $\varepsilon$ is not in $\Phi(L)$, because $\varepsilon \notin L$. With this, constructing the resulting DFA is now easy.