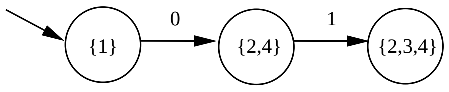
WS 2006/2007 Dr. C. Stamm / Roland Flury / Stefan Schmid

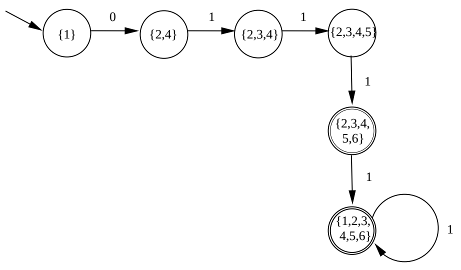
Discrete Event Systems
Solution to Exercise 3

1 Endliche Automaten und reguläre Sprachen [Exam!]

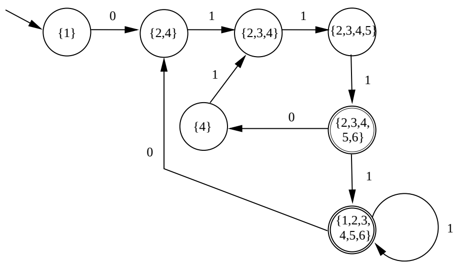
a) We could use the systematic transformation scheme presented in the lecture (slide 1/75). Considering the large number of states, however, this will easily lead to an explosion of states in the derandomized automaton. Hence, we build the deterministic finite automaton in a step-wise manner, only creating 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.



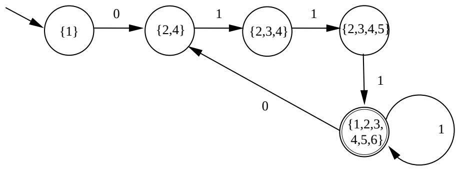
In any of the states 2, 3, and 4, only a 1 is accepted. Assume that the automaton is currently in state 2, this 1 can lead to states $\{2, 3, 4\}$ when including all ϵ transitions. When in state 3, the 1 leads to states $\{2, 3, 4\}$ and finally, when being in state 4, the reachable states given a 1 are $\{2, 3, 4, 5\}$. 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.



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:



It can easily be seen, however, that the states $\{4\}$, $\{2, 4\}$ and $\{2, 3, 4, 5, 6\}$, $\{1, 2, 3, 4, 5, 6\}$can be merged and hence, the automaton can be reduced to the one shown in the next Figure.



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

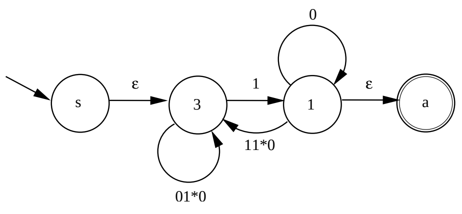
2 Pumping Lemma? [Exam!]

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 s to be the string 0110^p1^p. Because s is a member of L_1 and has length more than p, the pumping lemma guarantees that s can be split into three parts, $s = xyz$, where $|xy| \leq p$ and for any $i \geq 0$ the string xy^iz is in L_1. In order to obtain the contradiction, we must prove that for every possible splitting into three parts $s = xyz$ where $|xy| \leq p$, the string s cannot be pumped. We therefore consider the various cases. (1) If y consists of only the initial 0, only the initial 1, or a combination thereof, the string cannot be pumped without violating either the constraints $a = 1$ or $b = 2$. (2) Assume that y consists of only 0's from the second block. In this case, the string yyz has more 0's than 1's and hence $c \neq d$. (3) If y is of the form 10^*, the string $xyyz$ cannot be in L_1 anymore, either.

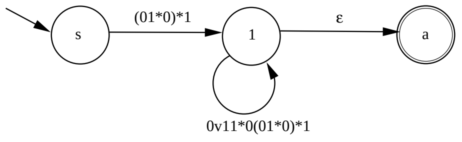
b) With the adapted language L_2, the proof of non-regularity is much more tricky! Specifically, non-regularity of L_2 cannot be proven using the pumping lemma, because any string in L_2 can actually be pumped! Consider for instance a string s of the form 0110^p1^p. In this case, we can split s into 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 string xy^iz 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 string that starts with the input-sequence 0110 is only accepted if the remainder of the string has the form $0^{c-1}d^c$ for some integer $c > 0$. Let s_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 s1. By the definition of A, this adapted finite automaton A' accepts all strings 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.

3 Automatentransformation [Exam!]

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 2, the corresponding GNFA looks like this:



After also removing state 3, the GNFA looks as follows.



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

b) The best way to solve this problem is to ask, which strings are actually not in $\Phi(L)$. The string 1, for instance must be in $\Phi(L)$, because the string 10 is in L. Moreover, the string 11 is in $\Phi(L)$, because 1101 is in L. Also, 10, 01, and 00 are in $\Phi(L)$ because of the strings 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 strings of length at least 2 are in $\Phi(L)$. Also, as seen before, the string 1 is in $\Phi(L)$. The only string that is not in $\Phi(L)$ is therefore 0, because there is no string of length 2 starting with 0 that leads to an accepting state. With this, constructing the resulting DFA is now easy.