

# CS 475: Formal Models of Computation

## Homework 2

1. Are the following languages regular? (Prove your answer):

- $L_1 = \{0^p 1^q 0^{p+q} \in \Sigma^* \mid p \geq 0, q \geq 0\}$

**[Answer].** One can prove that a word such as  $0^n 1 0^{n+1} \in L_1$  does not satisfy the pumping lemma. Alternatively, consider  $L'_1 = L_1 \cap 1^* 0^* = \{1^q 0^q \mid q \in \mathbb{N}\}$ .  $L'_1$  is a known non-regular language, which we obtained from  $L_1$  by a regularity preserving operation (intersection with the regular language  $1^* 0^*$ ). Regular languages are closed under intersection, which implies  $L_1$  is non-regular.

- $L_2 = \{0^{\binom{n}{2}} \in \Sigma^* \mid n \geq 2\}$

**[Answer].** If  $L_2$  is regular, then every string in  $L_2$  should satisfy the pumping lemma. Let  $p$  be the pumping length, and consider  $w = 0^{\binom{p+1}{2}} = xyz$ , with  $x = 0^l$ ,  $y = 0^m$ , and  $z = 0^{\binom{p+1}{2} - l - m}$ , with  $m > 0$ , and  $l + m \leq p$ . By the pumping lemma,  $xy^2z = 0^l 0^{2m} 0^{\binom{p+1}{2} - l - m} \in L_2$ . Since

$$\binom{p+1}{2} < \binom{p+1}{2} + l < \binom{p+2}{2},$$

$xy^2z \notin L_2$ , which implies  $L_2$  is not regular.

- $L_3 = \{w \in \Sigma^* \mid w = xzx^R \text{ for } x, z \in \{0, 1\}^*\}$

**[Answer].**  $L_3$  could alternatively be written with the regular expression  $0\{0, 1\}\{0, 1\}^*0 \cup 1\{0, 1\}\{0, 1\}^*1$ , therefore it is regular. Note that by making  $x$  a single character, the DFA must only remember the first character read.

2. If  $L$  is a regular language, are the following languages regular? (Prove your answer):

- $\text{PER}(L) = \{x_1 x_2 x_3 \in \Sigma^* \mid x_3 x_2 x_1 \in L, x_1, x_2, x_3 \in \Sigma^*\}$

**[Answer].** Let  $M = (Q, \Sigma, \delta, q_{start}, F)$  be the machine that accepts  $L$ . From  $M$  we are going to construct three types of machines, that together, accept  $\text{PER}(L)$ :

- For the first type, we define a machine  $A_q$  for every  $q \in Q$ , as  $A_q = (Q, \Sigma, \delta_{A_q}, q, \emptyset)$ . The function  $\delta_{A_q}$  is the function  $\delta$  extended such that for every state  $f \in F$ , there is an epsilon transition to the starting states of the machines  $B_{p,q}$ , which are defined next.
- The second type, for every pair of states  $p, q \in Q$ , define  $B_{p,q} = (Q, \Sigma, \delta_{B_{p,q}}, p, \emptyset)$ . As before,  $\delta_{B_{p,q}}$  is the function  $\delta$  extended such that there is an epsilon transition from the state  $q$  to the starting state of the machine  $C_p$ .
- For the third type, for every state  $p \in Q$ , define  $C_p = (Q, \Sigma, \delta, q_{start}, \{p\})$ .

Finally, we define a “global” start state, with epsilon transitions to the start state of each of the  $A_q$  machines. The intuition behind this construction is that if  $x_1x_2x_3 \in \text{PER}(L)$ , then  $x_1$  takes a path from some state  $q \in Q$  to a state in  $F$ . This is verified by the machines  $A_q$ . For this to work, however, we have to guarantee that  $q$  is reachable by reading  $x_2$  from some state  $p$ , which is the task of machine  $Bp, q$ . Finally, we need to check if  $p$  is reachable from the original start state by reading  $x_3$ .

- $\text{EVEN}(L) = \{w \in \Sigma^* \mid x_1, x_2, \dots, x_{2k} \in \Sigma, x_1x_2x_3x_4, \dots, x_{2k-1}x_{2k} \in L, \text{ and } w = x_2x_4x_6x_8 \dots x_{2k}\}$

The intuition here is that we need to somehow remove every character at an odd position. Inverse homomorphisms turn out to be very helpful for this particular task. First define the homomorphism  $h_1 : \{0, 1, c, \epsilon\} \rightarrow \{0, 1, \epsilon\}$ , with  $h_1(0) = 0$ ,  $h_1(1) = 1$ , and  $h_1(\epsilon) = c$ . We have:

$$L_1 = h_1^{-1}(L) = \{c^*a_1c^*a_2c^*a_3c^* \dots c^*a_nc^* \mid a_1a_2a_3 \dots a_n \in L\}.$$

Now consider

$$L_2 = L_1 \cap (c(0 \cup 1)c(0 \cup 1))^* = \{ca_1ca_2ca_3 \dots ca_n \mid a_1a_2a_3 \dots a_n \in L, n \geq 0, n \equiv 0 \pmod{2}\}.$$

Note that if we “ignore”  $c$ , then  $L_2$  contains all even length strings of  $L$ . Now we need to somehow distinguish even from odd positions. For this, we define a second homomorphism  $h_2 : \{0, 1, o, e, \epsilon\} \rightarrow \{0, 1, c, \epsilon\}$ , as  $h_2(0) = 0$ ,  $h_2(1) = 1$ , and  $h_2(o) = h_2(e) = c$ . Computing the inverse homomorphism  $h_2^{-1}(L_2)$ :

$$L_3 = h_2^{-1}(L_2) = \{(o \cup e)a_1(o \cup e)a_2(o \cup e)a_3 \dots (o \cup e)a_n \mid a_1a_2a_3 \dots a_n \in L, n \geq 0, n \equiv 0 \pmod{2}\}.$$

and considering the language  $(o(0 \cup 1)e(0 \cup 1))^*$ , compute the intersection:

$$L_4 = L_3 \cap (o(0 \cup 1)e(0 \cup 1))^* = \{oa_1ea_2oa_3 \dots ea_n \mid a_1a_2a_3 \dots a_n \in L, n \geq 0, n \equiv 0 \pmod{2}\}.$$

From  $L_4$  we have to “eliminate” all character pairs of the form  $oa_i$ , and also eliminate all  $e$  characters. For this purpose, we define (again...) a homomorphism  $h_3 : \{0, 1, 2, 3, \epsilon\} \rightarrow \{e0, e1, o0, o1, \epsilon\}$ , with  $h_3(0) = e0$ ,  $h_3(1) = e1$ ,  $h_3(2) = o0$ , and  $h_3(3) = o1$ . Computing the inverse homomorphism  $h_3^{-1}(L_4)$ :

$$L_5 = h_3^{-1}(L_4) \subseteq ((2 \cup 3)^+(0 \cup 1)^+(2 \cup 3)^+(0 \cup 1)^+)^*.$$

Intuitively,  $L_5$  is similar to  $L$ , only that 0s and 1s in odd positions are mapped to 2s and 3s respectively. Finally, define the homomorphism  $h_4 : \{0, 1, 2, 3\} \rightarrow \{0, 1, \epsilon\}$ , as  $h_4(0) = 0$ ,  $h_4(1) = 1$ ,  $h_4(2) = \epsilon$ , and  $h_4(3) = \epsilon$ , and apply it to  $L_5$ :

$$\text{EVEN}(L) = h_4(L_5).$$

Since we use only regularity preserving operations starting with  $L$ , this implies that  $\text{EVEN}(L)$  is regular.

- $\text{SQRT}(L) = \{x \in \Sigma^* \mid xy \in L, y \in \Sigma^*, \text{ and } |y| = |x|^2\}$

We construct an automaton  $M'$  that accepts  $\text{SQRT}(L)$ . First we are going to design a machine that *magically* guesses a correct partition of the input string into  $xy$ . First, let  $M(L) = (Q, \Sigma, \delta, q_0, F)$ . For  $q \in Q$ , and  $f \in F$ , define the automaton  $A_{q,f}$ . The guessing comes here with  $\hat{\delta}(q_0, x) = q$ , and  $\hat{\delta}(q, y) = f$ . In other words, before we read the whole input string, we guess that after reading  $x$ , the state is going to be  $q$ , and that after reading  $y$  from  $q$ , the final state is going to be  $f$ . We can do this for every pair of states  $q \in Q$ , and  $f \in F$ , therefore an NFA that runs all the  $A_{q,f}$  in parallel exists.

What remains is somehow to measure that  $|y| = |x|^2$ . In a particular machine  $A_{q,f}$ , we think of this as follows: for every *step* taken from  $q_0$ ,  $|x|$  steps have to be taken from  $q$ . The problem is that we do not know the length of  $x$ . To solve this, we define the set of states of  $A_{q,f}$  as:  $Q \times Q \times \text{pow}(QxQ) \times \text{pow}(QxQ)$ , with the transition equation  $\delta'((u, v, S, J), a) = (u', v', S', J')$ , such as:

- $u' = \delta(u, a)$ .
- $v'$  is any state in  $Q$ , selected nondeterministically. If we are lucky, it takes  $|x|$  steps to reach  $v'$  from  $v$ .
- $S' = \{(p, s') \in Q \times Q \mid \exists(p, s) \in S \text{ and } a \in \Sigma \text{ such that } \delta(s, a) = v'\}$ . In other words, after reading  $i$  characters of the input string,  $S'$  will contain all the pairs of states that are separated by  $i$  transitions.
- $J' = J \cup (v, v')$ . Here we just record all the nondeterministic jumps.

Define the initial state of  $A_{q,f}$  to be  $(q_0, q, \{(p, p) \mid \forall p \in Q\}, \emptyset)$ , and the accepting states to be all of the states of the form  $(q, f, S, J)$  with  $J \subseteq S$ . Note that after reading  $x$ ,  $S$  contains all the pairs of states separated by  $|x|$  transitions. If  $J \subset S$ , then that means that we guessed all the  $|x|$  jumps correctly, and that there is a  $y$  such that  $|y| = |x|^2$ .

- $\text{LOG}(L) = \{x \in \Sigma^* \mid xy \in L, y \in \Sigma^*, \text{ and } |y| = 2^{|x|}\}$

Let  $M(L) = (Q, \Sigma, \delta, q_0, F)$ . We construct automaton  $M' = (Q', \Sigma, \delta', q'_{start}, F')$  that decides  $\text{LOG}(L)$ , with:

- $Q' = Q \times \text{pow}(Q)^n$  with  $n = |Q|$ .
- $q'_{start} = (q_{start}, \{q_1\}, \{q_2\}, \dots, \{q_n\})$
- $F' = \{(q, S_1, S_2, \dots, S_n) \in Q' \mid q \in Q \text{ and } \exists f \in F, i \in \{1, 2, \dots, n\} \text{ such that } q, f \in S_i\}$ .

For the transition equation  $\delta'((q, S_1, S_2, \dots, S_n), a) = (q', S'_1, S'_2, \dots, S'_n)$ :

- $q' = \delta(q, a)$
- $S'_i = \bigcup_{q \in S_i} [S_j \text{ if } \exists a \in \Sigma \text{ such that } \delta(q, a) = q_j, \text{ or } \emptyset \text{ otherwise}]$ .

Intuitively, after reading  $m$  characters from the input string,  $S_i$  is the set of states reachable after following  $2^m$  transitions from  $q_i$ .

3. Let  $h$  be the homomorphism  $h(a) = 01$ ,  $h(b) = 0$ .

- Find  $h^{-1}(L_1)$ , where  $L_1 = (10 \cup 1)^*$ .

**Answer.** Remember that for any homomorphism  $h$ ,  $h(\epsilon) = \epsilon$ . The only word  $w \in (a^*b^*)^*$  such that  $h(w) \in L_1$  is  $w = \epsilon$ . Therefore,  $h^{-1}(L_1) = \{\epsilon\}$ .

- Find  $h(L_2)$ , where  $L_2 = (a \cup b)^*$ .

**Answer.**  $h(L_2) = (01 \cup 0)^*$ .

- Find  $h^{-1}(L_3)$ , where  $L_3$  is the set of all strings of 0's and 1's with an equal number of 0's and 1's.

**Answer.**  $h^{-1}(L_3) = a^*$ . (If a string  $w$  would contain  $b$ , then  $h(w)$  would have at least one more 0 than 1, and  $h(w) \notin L_3$ .)

4. Construct a language that is not regular, and that satisfies the pumping lemma.

**[Answer].** We have to pick a familiar example of a non-regular language and patch it up so that it satisfies the pumping lemma. The language consisting of strings of the form  $0^+1^k0^k$  is certainly non-regular. Next we have to patch it to satisfy the pumping lemma. The pumping lemma, roughly, says that a sub-string appearing near the first part of a sufficiently long string can be inserted into or deleted from the string and the resultant string will still be in the language. Let's pick the pumping length  $p$  to be 1 for simplicity. To patch things up, we expand our language by union it with the regular language  $1^*0^*$ . Now we prove that  $L = L_1 \cup L_2$ , with  $L_1 = \{0^+1^k0^k \in \Sigma^* \mid k \in \mathbb{N}\}$ , and  $L_2 = 1^*0^*$ , satisfies the pumping lemma:

We show that there exist a value of  $p$  such that the pumping lemma holds. In particular, let  $p = 1$ . Then we must show that for any string  $w \in L$  of length at least 1,  $w$  can be written  $xyz$ , with  $|xy| \leq p$ ,  $|y| \geq 1$ , and for all  $i \geq 0$ ,  $xy^iz \in L$ . The idea is that  $y$  will be simply the first character of  $w$ , and  $x$  will be the empty string. There are two cases:

- $w \in L_1$ , thus for some  $k$ , and  $j > 0$ ,  $w = 0^j1^k0^k$ . Let  $x = \epsilon$ ,  $y = 0$ , and  $z = 0^{j-1}1^k0^k$ . Now for any  $i > 0$ ,  $xy^iz = 0^{i+j-1}1^k0^k \in L_1 \subset L$ . Also, if  $i = 0$ , and  $j \geq 2$ ,  $xy^iz = xz = 0^{i+j-1}1^k0^k \in L_1 \subset L$ . On the other hand, if  $j = 1$ ,  $xy^iz = xz = 1^k0^k \in L_2 \subset L$ . Thus, regardless of  $i$ ,  $w = xy^iz \in L$ .
- $w \in L_2$ , thus  $w$  is of the form  $1^n0^m$ , for  $0 \leq n, m$ , which it is easy to show that satisfies the pumping lemma.

5. Find the minimum-state finite automaton equivalent to the transition diagram shown in Figure 1.

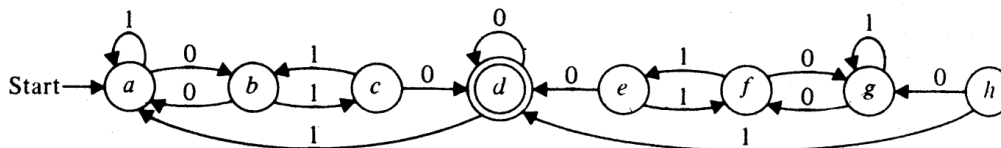


Figure 1.

First we remove the states  $e$ ,  $f$ ,  $g$ , and  $h$ , since they cannot be reached from the starting state  $a$ . For the rest of the states, we follow the algorithm presented in class:

(a) Construct the table:

b			
c			
d			
	a	b	c

(b)  $d$  is the only accepting state, therefore it can be distinguish from any other state:

b			
c			
d	X	X	X
	a	b	c

(c) Consider the pair  $(b, c)$ . After reading 0,  $c$  goes to  $d$ , which is accepting, but  $b$  goes to  $a$ , which is non-accepting. Therefore, we can distinguish between  $b$  and  $c$ .

b			
c		X	
d	X	X	X
	a	b	c

(d) Similarly, with the pair  $(a, c)$ , reading 0 takes  $c$  goes to  $d$ , but takes  $a$  to  $b$ . Since one is accepting, and the other is not, we can distinguish  $a$  and  $c$ .

b			
c	X	X	
d	X	X	X
	a	b	c

(e) Finally, for the pair  $(a, b)$ , reading 1 takes  $a$  to itself, and takes  $b$  to  $c$ . In the previous step we determined that  $a$  and  $c$  are distinguishable, therefore  $a$  and  $b$  are distinguishable.

b	X		
c	X	X	
d	X	X	X
	a	b	c

We conclude that after removing the unreachable states, the DFA is minimum.

6. Determine the Myhill-Nerode equivalence classes for  $L = \{w \in \Sigma^* \mid w \text{ has an equal number of 0s and 1s}\}$ . Use your answer to show that  $L$  is not regular.

**[Answer].** For  $w \in L$ , let  $\text{ZERO}(w)$  be the number of zeroes in  $w$ , and similarly, let  $\text{ONE}(w)$  be the number of ones in  $w$  (i.e., if  $w = 001010$ , then  $\text{ZERO}(w) = 4$ , and  $\text{ONE}(w) = 2$ ). Also, let  $Z_n = \{w \in L \mid \text{ZERO}(w) = n\}$ ,  $O_n = \{w \in L \mid \text{ONE}(w) = n\}$ , and  $D_k = \{w \in L \mid w \in Z_i, w \in O_j, \text{ and } k = i - j\}$ . For a complete answer, we would need to prove that the set of equivalence classes for  $L$  is  $\{D_k \mid k \in \mathbb{Z}\}$ .

7. Consider the input string for a DFA. A useful abstraction is to think of the input string as written, from left to right, on a *tape*. In this abstraction, the DFA has a *reader head*, that reads the characters written on the tape. The reader head starts at the leftmost character written on the tape, and as it moves to the right, it reads the rest of the characters of the input string.

- A 2DFA is a DFA that is able to move the reader head to the left and to the right (not only to the right, as a DFA). Prove that every for every 2DFA, there is a standard DFA that recognizes the same language.

**[Answer.]** The main insight here is that for a string  $w = xy$  accepted by the 2DFA  $M = (Q, \Sigma, \delta, q_{start}, F)$ ,  $M$  may cross a finite number of times the boundary between  $x$  and  $y$ . Define a function from  $T_x \mid Q \rightarrow Q \cup \emptyset$ , such that:

- $T_x(q_{start}) = q$ , for  $q \in \hat{\delta}(q_{start}, x)$ . That is, the state  $M$  reaches after reading  $x$ .
- When  $M$  backtracks, and crosses the boundary from  $y$  to  $x$ , let  $p \in Q$  be the last state  $M$  was in before going from  $y$  to  $x$ . If  $M$  comes back from  $x$  to  $y$ , with  $q$  being the last state  $M$  was in before going from  $x$  to  $y$ , define  $T_x(p) = q$ .
- Define  $T_x(p) = \emptyset$  if  $T_x(p)$  for  $p \in Q$  has not been defined in the previous two cases.

Consider a  $w' = x'y$  such that  $T_x(p) = T_{x'}(p), \forall p \in Q$ . Once defined, we note that  $T_x$  depends only on  $x$ . Moreover,  $M$  accepts  $w$  if and only if accepts  $w'$ . This follows because either  $M$  does not come back from  $y$  to  $x$ , and acceptance depends on  $y$ , or it does come back, but  $x$  and  $x'$  are indistinguishable given the assumption  $T_x(p) = T_{x'}(p)$ . In other words,  $x$  and  $x'$  belong to the same equivalence class given  $T_x(p) = T_{x'}(p)$ . Finally, the domain and codomain of  $T_x$  are finite, which means there is only a finite number of possible  $T_x$  functions. This implies there is a finite number of equivalence classes of strings for the language accepted by  $M$ , which means the language is regular. This proves that for every 2DFA there is DFA that accepts the same language. Since a DFA can be trivially converted to a 2DFA, the classes of DFAs and 2DFAs are equivalent.

- A one-pebble 2DFA is a 2DFA with the added capability of marking a character on the tape by placing a pebble on it. The transition equation depends on the present state, the tape character scanned, and the presence or absence of a pebble on the tape squared scanned. The output of the transition equation indicates the next state, a direction of motion for the reader head, and possibly placing or removing the pebble from the scanned character. The automaton rejects a string if it attempts to place a second pebble on the tape. Prove that for every one-pebble 2DFA, there is a DFA that recognizes the same language.

**[Answer.]** Assume  $M = (Q, \Sigma, \delta, q_{start}, F)$  is a one-pebble DFA. For this problem, we augment the input tape with two additional tapes, such that every character in the input is now a triplet  $(a, S_l, S_r)$ , such that  $a \in \Sigma$ , and  $S_l, S_r \in \text{pow}(QxQ)$ . The idea is to show that a one-pebble DFA in this augmented input never uses the pebble, and then to use a homomorphism to get rid of the extra input. A pair  $(p, q) \in S_l$ , indicates that from a give position of the tape, if the 2DFA were in state  $p$ , and it would move its head to the left, then under the assumption that it did not encounter the pebble, it would return to this position of the tape in state  $q$ . Similarly for  $(p, q) \in S_r$ , but for the head moving to the right. The extra tapes can be constructed using a one-pebble 2DFA, that for each position on the tape, it drops the pebble and then applies iteratively  $\delta$ , until the pebble is found again.

Now, suppose that  $M$  is in state  $p$ ,  $(p, q) \in S_r$  wants to drop the pebble, and then move the head to the right. Then, instead of dropping the pebble, we can immediately jump to  $q$ . Note that if there is not a  $(p, q)$ , then the pebble is not retrieved again, in which case it was not necessary to drop it. We have shown that there is a 2DFA that accepts strings on characters of the form  $(a, S_l, S_r)$ . Given the previous point, this language is regular. Defining a homomorphism  $h((a, S_l, S_r)) = a$ , gives the original language accepted by the one-pebble

DFA. Since regular languages are closed under homomorphism, and a DFA can be trivially converted to a one-pebble DFA, the classes of one-pebble DFAs and DFAs are equivalent.