

## Problem 1

Which of the following statements are correct? If you think a statement is correct, give a proof. If you think it is incorrect, give a counterexample.

- (a) If  $L$  is regular, then  $L' = \{w : w \text{ is a string in } L \text{ of odd length}\}$  is also regular.
- (b) If  $L$  is regular, then  $L' = \{wxw : w, x \text{ are strings in } L\}$  is also regular.
- (c) For a string  $w = w_1w_2 \dots w_n$ , let  $w^{\&e} = w_1w_1w_2w_2 \dots w_nw_n$ . For example,  $(abb)^{\&e} = aabbbb$ . If  $L$  is regular, then  $L' = \{w^{\&e} : w \in L\}$  is also regular.
- (d) If  $L$  is regular over alphabet  $\Sigma = \{0, 1\}$ , then  $L' = \{w : 0w1 \text{ is a string in } L\}$  is also regular.
- (e) If  $L$  is regular, then

$$L' = \{ww' : w, w' \text{ are strings in } L \text{ such that } w \neq w'\}$$

is also regular.

- (f) **(Extra credit)** For a string  $w$ , let  $w^{\text{odd}}$  be the string obtained by keeping only those symbols of  $w$  in odd positions. For instance, if  $w = \text{springs}$ , then  $w^{\text{odd}} = \text{srns}$ ; if  $w = \text{mama}$ , then  $w^{\text{odd}} = \text{mm}$ . If  $L$  is regular, then

$$L' = \{w^{\text{odd}} : w \text{ is a string in } L\}$$

is also regular.

## Solution

- (a) Correct. We can write  $L'$  as the intersection of  $L$  and the language of all strings of odd length (which is also regular – we can represent it by the regular expression  $(a_1 + \dots + a_k)((a_1 + \dots + a_k)(a_1 + \dots + a_k))^*$ , where  $\{a_1, \dots, a_k\}$  is the alphabet of  $L$ ). Since regular languages are closed under intersection,  $L'$  must be regular.
- (b) Wrong. For example, let  $\Sigma = \{a, b\}$  and  $L = \{a^n b : n \geq 0\}$ . Then

$$L' = \{a^n b a^m b a^n b : m, n \geq 0\}$$

It is easy to see that  $L$  is regular (it is described by the regular expression  $a^*b$ ), but  $L'$  is not. We prove this using the pumping lemma. Consider an arbitrary pumping length  $n$ . Let  $x = a^n b b a^n b$ , which is a string in  $L'$ . Now for any partition  $x = uvw$ , where  $|uv| \leq n$  and  $v \neq \varepsilon$ , the string  $wv^2w$  is of the form  $a^k b b a^n b$ , where  $k > n$ . This string is not in  $L'$ : It has more  $a$ s in the first part than in the last part.

- (c) Correct. Given a DFA  $M$  for  $L$ , we can construct a DFA  $M'$  for  $L'$ . Informally, we split every transition of  $M$  with an intermediate state. So every transition of  $M$  yields two consecutive transitions in  $M'$ . If the transition in  $M$  was labeled by a symbol  $a$ , then the two corresponding transitions in  $M'$  will be labeled by  $a$ .

Formally, the states of  $M'$  will include all the states of  $M$ , plus additional states  $q_e$ , one for every transition  $e$  of  $M$ . The start and accept states in  $M'$  are the same as those in  $M$ . For every transition  $e$  from  $q$  to  $q'$  on input  $a$ , we include a pair of transitions in  $M'$ : One is a transition from  $q$  to  $q_e$  on input  $a$ , and the other one is a transition from  $q_e$  to  $q'$  on input  $a$ . (Technically, this gives an NFA because the states  $q_e$  are missing some outgoing transitions, but we can easily convert this NFA to a DFA by adding a “die” state, or using the generic method from class.)

Alternatively, we can take a regular expression for  $L$  and convert it into one for  $L'$ : Simply replace every symbol in  $\Sigma$  with a pair of symbols. This is compatible with the rules for generating regular expressions:  $\emptyset^\& = \emptyset$ ,  $\varepsilon^\& = \varepsilon$ ,  $a^\& = aa$  for every alphabet symbol  $a$ ,  $(R_1R_2)^\& = R_1^\&R_2^\&$ ,  $(R_1 + R_2)^\& = R_1^\& + R_2^\&$ , and  $(R^*)^\& = (R^\&)^*$ . So we can perform the desired operation  $\&$  on the regular expression for  $L$  and obtain a regular expression for  $L'$ .

- (d) Correct. First, if  $L$  is regular, then the intersection of  $L$  and the language described by the regular expression  $0(0 + 1)^*1$  is also regular. Let  $M = (Q, \Sigma, \delta, q_0, F)$  be a DFA for this language. Given  $M$ , we can build a DFA  $M'$  for  $L'$ .  $M'$  has the same states and transitions as  $M$ , but different start and accept states. The start state of  $M'$  is the state that is reached from  $q_0$  on a 0. The accept states of  $M'$  are those states that reach accept states of  $M$  on a 1. Formally,  $M' = (Q, \Sigma, \delta, q'_0, F')$ , where  $q'_0 = \delta(q_0, 0)$ , and  $F' = \{q : \delta(q, 1) \in F\}$ .
- (e) Wrong. Let  $\Sigma = \{a, b\}$  and  $L = \{a^n b : n \geq 0\}$ . Then

$$L' = \{a^m b a^n b : m \neq n\}.$$

We will argue that  $L'$  is not regular. It is possible to do so directly, but in fact it is easier to prove that its complement  $\overline{L'}$  is not regular instead. Since complementation preserves regularity, it follows that  $L'$  cannot be regular.

We prove  $\overline{L'}$  is not regular via the pumping lemma. Let  $n$  be the pumping length. Then the string  $a^n b a^n b$  is in  $\overline{L'}$ . Now for any partition  $x = uvw$ , where  $|uv| \leq n$  and  $v \neq \varepsilon$ , the string  $uv^2w$  is of the form  $a^m b a^n$ , where  $m > n$ . This string is in  $L'$ , so it is not in  $\overline{L'}$ . By the pumping lemma,  $\overline{L'}$  is not regular, so  $L'$  is not regular either.

- (f) Correct. Let  $\Sigma = \{a_1, \dots, a_k\}$  be the alphabet of  $L$ . Without loss of generality, assume  $\Sigma$  does not contain the special symbol  $\star$ . Let  $\Sigma^\star = \Sigma \cup \{\star\}$ .

Consider the following transformation on a string  $w \in \Sigma^\star$ . You are to replace any symbol in  $w$  with the special symbol  $\star$ . For instance, starting from the string  $\mathbf{me}$ , using this operation you can derive the strings  $\mathbf{m\star}$ ,  $\mathbf{\star e}$ , and  $\mathbf{\star\star}$ . Let  $L_1$  be the language obtained by applying this transformation on all strings in  $L$ .

We claim that if  $L$  is regular, then  $L_1$  is also regular. To see this, take any regular expression for  $L$ , and replace every occurrence of an alphabet symbol  $a \in \Sigma$  by the expression  $(a + \star)$ . This gives you a regular expression for  $L_1$ . Alternatively, take a DFA for  $L$ , and for every existing transition, add another one labeled by  $\star$ . This gives an NFA for  $L_1$ .

Now consider the intersection of  $L_1$  and the language described by the regular expression  $((a_1 + \dots + a_k)\star)^*$ . Call this language  $L_2$ . Then every string in  $L_2$  is obtained from a string in  $L$  by replacing the symbols in even positions by  $\star$ . For example, if  $me$  is in  $L$ , out of the four strings that arose from it in  $L_1$ , only the string  $m\star$  is in  $L_2$ .

Since  $L_2$  is the intersection of regular languages, it is also regular. Now let  $M$  be a DFA for  $L_2$ . Replace all the transitions labeled by  $\star$  in  $M$  by  $\varepsilon$ -transitions. This transformation “kills” all the  $\star$ s in  $L_2$ . The resulting NFA will be an NFA for  $L'$ . Therefore  $L'$  is regular.

## Problem 2

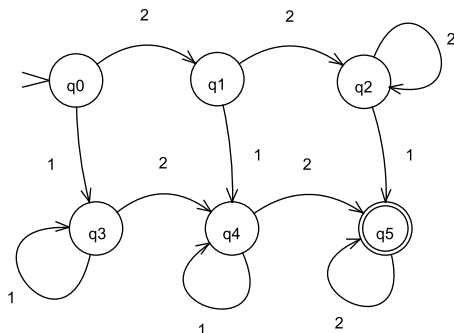
Which of the following languages are regular, and which are not? If a language is regular, draw the minimal DFA for it and show that every pair of states is distinguishable. If a language is not regular, prove it using the pumping lemma.

- (a)  $L_1 = \{0^n 10^n : n \geq 0\}$ ,  $\Sigma = \{0, 1\}$ .
- (b)  $L_2 = \{0^n 10^m : m \neq n; m, n \geq 0\}$ ,  $\Sigma = \{0, 1\}$ .
- (c)  $L_3 = \{w : w \text{ has fewer 0s than 1s and 2s together}\}$ ,  $\Sigma = \{0, 1, 2\}$ .
- (d)  $L_4 = \{w : w \text{ has at least one 1 and at least two 2s}\}$ ,  $\Sigma = \{1, 2\}$ .
- (e)  $L_5 = \{w : w \text{ has the same number of patterns 011 and 110}\}$ ,  $\Sigma = \{0, 1\}$ .
- (f)  $L_6 = \{w : w \text{ has the same number of patterns 010 and 101}\}$ ,  $\Sigma = \{0, 1\}$ .

## Solution

- (a)  $L_1$  is not regular. Assume otherwise and let  $n$  be the pumping length for  $L_1$ . We consider the string  $z = 0^n 10^n \in L_1$ . No matter how we break it up into  $uvw$  where  $|uv| \leq n$  and  $|v| > 0$ ,  $v$  consists only of some 0s in the initial block. Therefore,  $uv^2w = 0^m 10^n$ , where  $m > n$ . This string is not in  $L_1$ . By the pumping lemma,  $L_1$  cannot be regular.
- (b)  $L_2$  is not regular. This can be shown directly, but it is easier to prove that  $\overline{L_2}$  is not regular instead. Since regular languages are closed under complement, it will follow that  $L_2$  cannot be regular either.  
 Assume otherwise and let  $n$  be the pumping length for  $\overline{L_2}$ . We now use the same example as in part (a):  $z = 0^n 10^n \in L_2$ , but no matter how we break up  $z$  as  $uvw$ , where  $|uv| \leq n$  and  $|v| > 0$ , we have  $uv^2w = 0^m 10^n$ ,  $m > n$ . This string is in  $L_2$ , that is outside  $\overline{L_2}$ , so  $\overline{L_2}$  cannot be regular by the pumping lemma.
- (c)  $L_3$  is not regular. Assume otherwise and let  $n$  be the pumping length for  $L_3$ . Choose  $z = 0^{2n} 1^n 2^{n+1}$ , so that  $z \in L_3$ . Now, no matter how we break up  $z$  as  $uvw$ , where  $|uv| \leq n$  and  $|v| > 0$ , we have  $uv^2w = 0^m 1^n 2^{n+1}$ , where  $m \geq 2n + 1$ . So  $uv^2w$  has more 0s than 1s and 2s together, and it is not in  $L_3$ . By the pumping lemma,  $L_3$  cannot be regular.

(d)  $L_4$  is regular. Consider the following DFA:

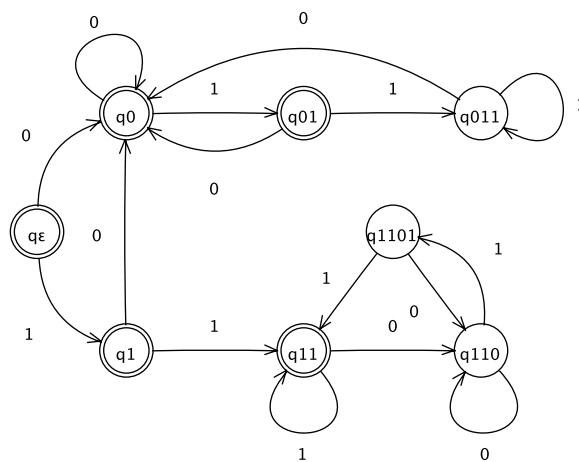


This DFA also has 0-loops around every state, which we omitted for clarity. The state in this DFA remembers how many 1s and 2s we have seen so far. Once we have reached our goal of one 1 and two 2s, we do not need to remember any additional ones.

We can see the DFA is minimal from the following table, which gives a pair of distinguishable strings for every pair of states:

$q_1$	21				
$q_2$	1	1			
$q_3$	22	22	1		
$q_4$	2	2	2	2	
$q_5$	$\varepsilon$	$\varepsilon$	$\varepsilon$	$\varepsilon$	$\varepsilon$
	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$

(e)  $L_5$  is regular. Consider the following DFA:



In this DFA, the state  $q_{011}$  indicates an excess of patterns 011, and the state  $q_{110}$  indicates an excess of patterns 110. The main idea is that in each string, the patterns 011 and 110 must alternate. The DFA keeps track of which is the last pattern seen.

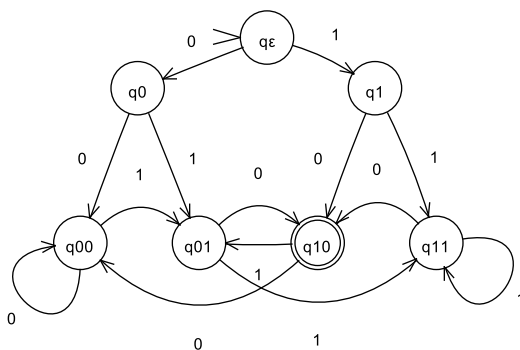
To show every pair of states is distinguishable, we give a string that distinguishes them in the following table:

$q_0$	11						
$q_{01}$	1	1					
$q_{011}$	$\varepsilon$	$\varepsilon$	$\varepsilon$				
$q_1$	10	11	1	1	$\varepsilon$		
$q_{11}$	0	0	0	$\varepsilon$	0		
$q_{110}$	$\varepsilon$	$\varepsilon$	$\varepsilon$	11	$\varepsilon$	$\varepsilon$	
$q_{1101}$	$\varepsilon$	$\varepsilon$	$\varepsilon$	0	$\varepsilon$	$\varepsilon$	1
	$q_\varepsilon$	$q_0$	$q_{01}$	$q_{011}$	$q_1$	$q_{11}$	$q_{110}$

- (f)  $L_6$  is not regular. Assume otherwise and let  $n$  be the pumping length. Let  $z = (010)^n 01(101)^n$ . This string is in  $L_6$ : It has exactly  $n$  patterns 010 and  $n$  patterns 101. No matter how we break up  $z$  into  $uvw$  where  $|uv| \leq n$  and  $|v| > 0$ ,  $v$  is in the first part of string  $z$ . Now we consider two cases. If  $v$  is not the string 0, then pumping  $v$  out will destroy at least one 010. If  $v = 0$ , then pumping it out will change the pattern  $\dots 010\underline{0}10\dots$  into  $\dots 01010\dots$ , thus it generates one more 101 pattern in the first half of string  $z$ . In either case, string  $uv$  has fewer 010 patterns than 101 patterns and doesn't belong to  $L_6$ . So  $L_6$  is not regular by the pumping lemma.

### Problem 3

This problem concerns the following DFA.



- (a) Run the minimization algorithm on this DFA. Clearly show the different stages that the minimization algorithm goes through.
- (b) Show that every pair of states in the minimized DFA is distinguishable.

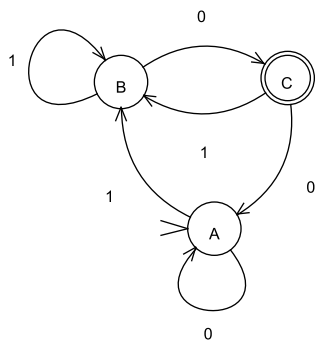
- (c) Convert the minimized DFA into a regular expression using the conversion algorithm from class.

## Solution

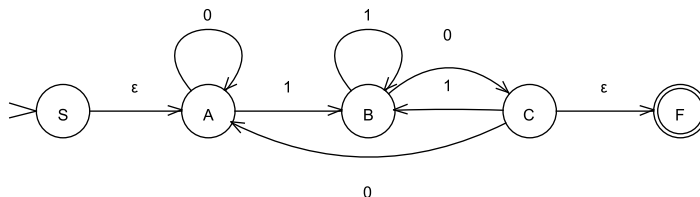
- (a) To obtain the minimized DFA, we mark all pairs of distinguishable states. First, every state (but  $q_{10}$ ) is distinguishable from  $q_{10}$ . Then, on transition 0, we note that  $q_1$ ,  $q_{01}$ , and  $q_{11}$  go to  $q_{10}$ , while  $q_\varepsilon$ ,  $q_0$ , and  $q_{00}$  go to a state distinguishable from  $q_{10}$ . So these are all distinguishable pairs. At this point we have the following pairs of distinguishable states:

$q_0$						
$q_1$	x	x				
$q_{00}$			x			
$q_{01}$	x	x		x		
$q_{10}$	x	x	x	x	x	
$q_{11}$	x	x		x		x
	$q_\varepsilon$	$q_0$	$q_1$	$q_{00}$	$q_{01}$	$q_{10}$

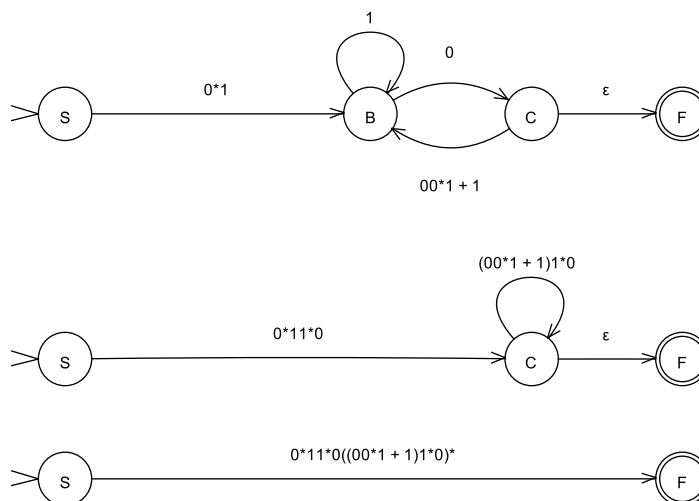
No more pairs of states can be distinguished. The indistinguishable states split into three classes: Class *A* consisting of states  $\{q_\varepsilon, q_0, q_{00}\}$ ; class *B*, consisting of  $\{q_1, q_{01}, q_{11}\}$ , and class *C*, consisting of  $q_{10}$ . This yields the following minimized DFA:



- (b) States *A* and *B* can be distinguished by 0; *A* and *C* can be distinguished by  $\varepsilon$ ; and *B* and *C* can be distinguished by  $\varepsilon$ .
- (c) We turn the DFA into a regular expression in successive stages. First, we add a new start state *S* that is not pointed to, and a new accept state *F* that does not point to any other state.



We now eliminate states  $A$ ,  $B$ , and  $C$  in succession.



We obtain the regular expression  $0^*11^*0((00^*1 + 1)1^*0)^*$ . It represents all strings that end in 00, albeit in a roundabout way. To see this, notice that the initial pattern  $0^*11^*0$  represents the first part of the input that ends in 10. The starred expression  $(00^*1 + 1)1^*0$  represents every subsequent (minimal) part that ends in 10. Each such part consists of some zeros ending with a 1 (this is the expression  $00^*1 + 1$ ), possibly followed by more 1s and then a 0.

## Problem 4

You have a file `cities.txt` that lists the latitudes, longitudes, and time zones of various cities around the world:

Aberdeen, UK	57.4N	2.9W	12.00pm
Adelaide, Australia	34.55S	138.36E	9.30pm
Algiers, Algeria	36.50N	3E	1.00pm
Ankara, Turkey	39.55N	32.55E	2.00pm
Ascuncion, Paraguay	25.15S	57.40W	8.00am
...			

Write `grep` commands (including a short explanation about how they work) that will search for the following information in the file:

- All cities in the US and Canada in the 6.00am time zone.
- All cities in the tropics (latitude 33S to 33N).
- All cities with longitude between 10E and 20E *not* in the 1.00pm time zone.

(d) All cities whose time zone does not fall on the hour, for example:

Adelaide, Australia	34.55S	138.36E	9.30pm
Kathmandu, Nepal	27.43N	25.19E	5.45pm
Rangoon, Myanmar	16.50N	96E	6.30pm

## Solution

(a) `grep '(US|Canada).*6\..00am' cities.txt`

This expression matches any (sub)string that starts with `US` or `Canada`, followed by any string, and ending with `6.00am`.

(b) `grep '((3[0-2]|0[0-9])\.[0-9]*)?|33(\.0*)?(S|N)' cities.txt`

To describe latitude between 0 and 33, we need to match every number in this range. This includes numbers 0 to 32, possibly followed by decimals, and the number 33, or 33 followed by a dot and some zeros. This is followed by `S` or `N`, which designates the latitude.

(c) `grep '((([1][0-9]\.[0-9]*)|20)E  
(\*am|(((0[2-9]|1[0-9]|2[0-4])\.[0-5][0-9])|1\.[1-5][0-9]|1\..0[1-9]))pm)'`  
`cities.txt`

We match the range 10E to 20E as in the last problem. To disqualify the 1.00pm time zone, we allow all am time zones, and all pm time zones that begin with anything other than 1, and those that begin with 1 but not followed by 00.

(d) `grep '(1?[0-9]|2?[0-4])\.(0[1-9]|[1-5][0-9])[ap]m' cities.txt`

`1?[0-9]` or `2?[0-4]` matches hour part and `0[1-9]` or `[1-5][0-9]` matches the minute part that does not fall on the hour.