Due: Tuesday, Dec. 3, at the beginnning of class

This is a group homework. Please staple your homework in four pairs. 1 and 2, 3 and 4, 5 and 6, 7 and 8. Hand in each of them to the corresponding stack in class.

1. **(10 pts) For the languages below, either prove that it is a regular language by giving an NFA or DFA that recognizes it; otherwise, argue that it is not a regular language. The alphabet is $\Sigma = \{0, 1\}$.**

   (a) **(10 pts) $L = \{w \mid w$ contains at least two 0's and at most one 1's$\}$**

   Solution: Transitions for a DFA that recognizes L are given below: ($\delta(x, i) = y$, means a transition to state y given an input i in state x . If an input is seen in a state but there's no transition function for it below the DFA will crash thus rejecting the input)

$$\delta(q_0, 0) = q_1, \delta(q_0, 1) = q_3$$

$$\delta(q_1, 0) = q_2, \delta(q_1, 1) = q_4$$

$$\delta(q_2, 0) = q_2, \delta(q_2, 1) = q_5$$

$$\delta(q_3, 0) = q_4$$

$$\delta(q_4, 0) = q_5$$

$$\delta(q_5, 0) = q_5$$

   Accept states are q_2 and q_5 ■

   (b) **(10 pts) $L = \{w \mid$ every block of 5 symbols of w contains at least two 1's$\}$**

   Solution (Based on cs375 hw): Even though there might be smaller DFAs for this problem we choose a pretty big one for the sake of simplicity of the proof. We label the states with bit strings. The label of a state will represent the last up to 5 symbols read. Thus for example if the last five symbols read are 11010, we will be in state 11010. The codes are at most 5 symbols long and exclude 10000, 01000, 00100, 00010, 00001, and 00000 (these are the unacceptable histories). The start state has label ϵ. All states are accept state. The transition function will be defined as follows

   - If s is a state code of length less than 5, then

$$\delta(s, 0) = s0$$

$$\delta(s, 1) = s1$$

   - For a state code s = $s_1s_2s_3s_4s_5$ of length 5,

$$\delta(s, 0) = s_2s_3s_4s_50$$

$$\delta(s, 1) = s_2s_3s_4s_51$$

     In this case, if $s_2s_3s_4s_50$ or $s_2s_3s_4s_51$ are among the excluded codes mentioned above, we omit the transition, allowing the DFA to crash (and thus reject) if a transition to these states are required.

   The correctness of this design is almost immediate. We simply keep track of the last 5 symbols read starting with no symbols at all, adding the last symbol read to the end of the state code and possibly dropping the first symbol of the code if the code becomes too long. The excluded codes are exactly those possible blocks of 5 consecutive symbols that have less than two 1's. Clearly, if a string has a block of length 5 with less than 2 one's, it will end up trapped in one of the excluded states. Also a string is never rejected unless it has a block of length 5 with no or only one 1's. ■

2. (10 pts) Use the same instructions and assumptions as for Problem 1, and solve the following problems.

(a) (10 pts) $L = \{w \mid w$ **contains an equal number of occurences of 0's and 1's**$\}$

Solution: Let $J = \{0^n 1^n \mid n = 0, 1, 2, \ldots\}$. J is made up of all strings consisting of a block of 0s followed by a block of an equal number of 1s. Thus J = $\{0^*1^*\} \bigcap$ L. Since J is known to be non-regular (proof on page 663 of Rosen text) but the intersection of two regular languages is regular, L can not be regular. ■

(b) (10 pts) $L = \{w \mid w$ **contains an equal number of occurences of the substring 01 and 10**$\}$

Solution (Based on cs375 hw): This means that for example 101 $\in$ L because 101 contains a single 01 and a single 10, but 1010 $\notin$ L because 1010 contains two 10s and one 01. Let k_w be the number of 01s in the string w, and let l_w be the number of 10s in the the string w. The crucial thing to observe is that for any string w, $\mid k_w - l_w \mid \leq 1$. Morever if $k_w > l_w$ then it must end in a 1, and if $l_w > k_w$ then it must end in a 0. Therefore what the DFA needs to remember, as it reads the input, is the following:

i. Nothing has been read so far
ii. String read so far has equal number of 01s and 10s but ends in a 0
iii. String read so far has equal number of 01s and 10s but ends in a 1
iv. String has one more 01 than 10
v. String has one more 10 than 01

So our DFA will have states corresponding to the above "equivalence classes". Let q_0 correspond to (i), q_1 to (ii), q_2 to (iii), q_3 to (iv), and q_4 to (v). The formal definition is as follows. The states are q_0, q_1, q_2, q_3, and q_4. The initial state is q_0 . The accept states are q_0, q_1, and q_2. The transition function is defined below.

$$\delta(q_0, 0) = q_1, \delta(q_0, 1) = q_2$$

$$\delta(q_1, 0) = q_1, \delta(q_1, 1) = q_3$$

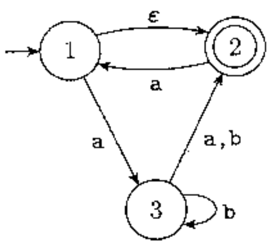
$$\delta(q_2, 0) = q_4, \delta(q_2, 1) = q_2$$

$$\delta(q_3, 0) = q_1, \delta(q_3, 1) = q_3$$

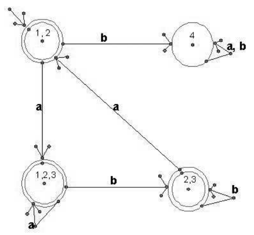
$$\delta(q_4, 0) = q_4, \delta(q_4, 1) = q_2$$

■

3. (10 pts) Convert the following NFA into a DFA that recognizes the same language.



Solution: Look at the graph below



■

4. (10 pts) Determine whether the following statements are true in general. They might be for specific cases, but are they always true? Justify your answer.

(a) If L_1L_2 is regular and $|L_1|$ is finite, then L_2 is regular

Solution: False. Let $L_1 = \emptyset$. Then now matter what L_2 is, $L_1L_2 = \emptyset$ and therefore is regular. ■

(b) If $L_1 \cup L_2$ is regular and L_1 is regular, then L_2 is regular

Solution: False. Take $L_1 = (0+1)^*$, and $L_2 = \{0^n1^n | n \geq 0\}$. Then $L_1 \cup L_2 = (0+1)^*$ is regular, and so is L_1, but L_2 is not. ■

5. (10 pts) Prove that by allowing multiple start states, any NFA with ϵ moves can be converted into an NFA that has no ϵ moves. The new NFA must use the same states as the original NFA.

Solution: The reason that ϵ moves can be reduced in NFA is that in NFA, transition function f can assign multiple states to a pair of state and input. If there is a ϵ move from state s_i to s_j, in the transition function, we add s_j as the destination state at any place where s_i is a destination function. For example, if there is a transition from state s_k to s_i with input a, we also add the move from state s_k to s_j with input a. If state s_i is a start state, we can set state s_j as start state too. (That's the reason why multiple start states are allowed in the problem.) Then the ϵ move from state s_i to s_j can be removed, and the new NFA works in the same way as the original one.

■

6. (10 pts) Give a brief description (mostly or completely in English) of the language described by each of the following regular experessions. The alphabet is given by $\Sigma = \{0, 1\}$.

(a) $(1 + \epsilon)(00^*1)^*0^*$

(b) $(\Sigma 01 \Sigma)^* \cup \emptyset$

(c) $((01) \cup (10))^*(111 \cup 1^*) \cup \epsilon$

Solution: (a) This regular expression generates all strings in which every 1 is separated by one or more 0. It also accepts the string which has only one 1 or the empty string.

(b) This set consists all strings that have zero or more 4-length substrings, each with 01 in the middle.

(c) This regular expression generates all strings consisting equal number of 0's and 1's at every position of even length from the beginning and with no more than two 0's or 1's consecutive, followed by zero or more 1's.

■

7. (10 pts) Design a Turing machine for the language $L = \{ww^R \mid w$ is any string of 0's and 1's$\}$, in which w^R denotes the reverse of w. Thus, the machine should decide whether the input is a palindrome. Describe the finite control of your machine in detail (possibly as a kind of pseudocode), but not necessarily with the specification of all the elements in the formal notation of a Turing machine.

Solution: We want a Turing machine that, starting at the leftmost position of the string, keeping the information of the symbol at the position and moving right to the rightmost position, comparing the symbol at that position. If they are same, the machine return back the left, compare the second leftmost symbol and second rightmost symbol, and so on. If at any position, a difference is found, the machine halts in a state that is not final state.

To build such a machine, we will use an auxiliary symbol M as a marker. The Turing machine successively replaces a symbol at the leftmost position of the string with an M and enters a state according to the symbol it read. (For example, if the machine read 0, it enters state s_0, if the machine read 1, it enters state s_1). Then the machine moves right staying in that state until it reaches the right most position. If the symbol at the rightmost position is 1, and the state of Turing machine is s_0, or the rightmost position has 0, while the state of the Turing machine is s_1, then the machine halts at a state which is not a final state. Otherwise, the symbol at the rightmost position of the string is replaced by an M, the Turing machine moves to left most position, the machine sweeps back and forth, terminates in a final state if and only if all the symbols are successively replaced.(The last symbol to be replaced should be at the right most position, as the machine starts from left most position)

■

8. (10 pts) Design a Turing machine (specified in the same way as in the previous problem) that decides in polynomial time membership in the language POW-PERM. The language is defined as follows. For the set $\{1, 2, 3, \ldots, k\}$, let p denote a permutation function. Furthermore, let p^t denote the composition of p with itself t times (for example, if p represents a circular left shift, then p^2 represents two circular left shifts). The language is defined as
POW-PERM =

$$\{\langle p, q, t\rangle \mid p = q^t \text{ in which p and q are permutations on } \{1, 2, \ldots, k\} \text{ and } t \text{ is a binary integer}\}$$

Note that the most obvious algorithm does not run in polynomial time. Hint: First try it for the case in which t is a power of 2.

Solution: let $t = \alpha_0 2^0 + \alpha_1 2^1 + \ldots + \alpha_n 2^n$, where α_i is either 0 or 1.
So $q^t = q^{\alpha_0 2^0 + \alpha_1 2^1 + \ldots + \alpha_n 2^n} = q^{\alpha_0 2^0} \times q^{\alpha_1 2^1} \times \ldots \times q^{\alpha_n 2^n}$

The trick is instead applying q t times on the set, we can apply q^{2^i} directly. It is easy to get the permutation form of q^{2^i}, just apply q on q itself, we can get q^2, apply q^2 on q^2, we can get q^4, and so on.....

Now we return to the Turing machine. The alphabet I of the machine is the set $\{1, 2, 3, \ldots, k\}$ (plus blank symbol B), the input of the machine is p,q,t, where both p and q have length k, t is a binary integer, represented as $t = \alpha_0 2^0 + \alpha_1 2^1 + \ldots + \alpha_n 2^n$.

Here is the algorithm:
1. Get an auxiliary space K with length k (which can be at the left to the original input or the right to the original input), initialize it as $\{1, 2, \ldots, k\}$, let $q_{aux} = q$
2. Scan the binary integer t in the order $\alpha_0, \alpha_1, \ldots, \alpha_n$. For each α_i, do step 3 and 4
3. If α_i is 0, do nothing on the K. Otherwise, apply q_{aux} on K.
4. Apply q_{aux} on itself, let $q_{aux} = q_{aux} \times q_{aux}$
5. Compare K, p to see whether they are same, if yes, this input is a member of POW-PERM.

■