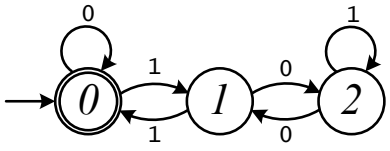
CS181 Winter 2015 - Problem Set 1 Solutions

1. **(10 points).** Give a DFA for the language

$$L = \{x \in \{0,1\}^* \mid x \text{ interpreted as a binary number is divisible by } 3\}.$$

As an example, the binary string 001011 translates to the decimal number 11 and is thus not divisible by 3. Here, we'll assume that the empty string ε translates to 0 and is in L. Please provide both a **formal description** and a **state diagram** of your machine, and make sure your machine is indeed a **D**FA. (*Hint: Appending '0' to the right of a binary number doubles it. Appending '1' to the right doubles it and adds 1.*)

Solution. The idea here is to keep track of the binary number mod 3 (*i.e.* the input's remainder after division by 3). In this case, the remainder can be one of three values: 0, 1 or 2. Thus, we use a state for each of these possible values. Notice if x has a remainder of r, then $x0$ has a remainder of $2r$ (mod 3). Thus, if our remainder is 0, then after appending a 0 the remainder is still 0! If our remainder is 1, then appending 0 makes the remainder 2. Similarly, if the remainder is 2, then appending 0 doubles the remainder to 4 but we can subtract out another 3 to get a final remainder of 1. Similar logic follows for appending a 1. Our final DFA is pictured below:



2. **(15 points).** Let L be any language, let L_R be the set of reversals of strings in L so that

$$L_R = \{x| \text{ for some } y \in L, |y| = |x| = n \text{ and } x_1x_2 \dots x_n = y_n y_{n-1} \dots y_1\}$$

Show that, if L is regular, so is L_R.

Solution. Let A be a regular language and $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA for A. Then consider the NFA $M^R = (Q \cup \{q_0^R\}, \Sigma, \delta^R, q_0^R, \{q_0\})$ where

$$\delta^R(q,a) = \begin{cases} F, & \text{if } q = q_0^R \text{ and } a = \varepsilon \\ \emptyset, & \text{if } q = q_0^R \text{ and } a \neq \varepsilon \\ \{p \in Q \mid \delta(p,a) = q\}, & \text{otherwise} \end{cases}.$$

Intuitively, this machine is constructed as follows: All the transitions of M are reversed and the initial states and final states are "swapped". However, this may yield multiple initial states which isn't allowed (by our definition). To fix this, we introduce a new initial state q_0^R which has ε-transitions to each of these states.

Let us quickly show why this works. Say w is accepted by M. Then M starts from q_0, follows some state sequence, then ends in a final state q_f. In M^R, if we read w backwards (i.e. read w^R), then an accepting run would jump first to q_f then follow the same state sequence backwards and wind up in q_0 (which is M^R's accepting state). Thus if $w \in L(M)$ then $w^R \in L(M^R)$. We can use the same argument to show that if $w \in L(M^R)$ then $w^R \in L(M)$. In particular, $L(M^R) = L(M)^R$.

3. **(15 points).** Let L be any language, let L_{alt} be the set of strings in L with their alternate characters removed, so that

$$L_{alt} = \{x| \text{ for some } y \in L, x_1x_2x_3 \ldots = y_1y_3y_5 \ldots\}$$

Show that, if L is regular, so is L_{alt}.

Solution. Here we would like to use a NFA to guess for us every other symbol when trying to accept a string x. We will need to do this in a manner such that $x_1x_2x_3 \ldots = y_1y_3y_5 \ldots$ and $y \in L$. To acheive this we perform the following conversion.

Let M be any DFA which accepts L. Consider the following new machine M'. We will use two copies of M to construct M'. Lets call the copies M_1 and M_2 and let the states $Q' = Q_1 \cup Q_2$.

The intuition is that M_1 will be represent states from which we make transitions of every input letter and M_2 will represent states from which we make only ϵ transitions denoting every missing letter.

To achieve this, first on any input alphabet, suppose we were in a state in M_1, we would like to transition to a state in M_2. Do this by converting the transition function as follows. Suppose we are in state $q_i^1 \in M_1$, $\delta(q_i^1, a) = q_j^1$. We take this transition out and add the following new transition $\delta(q_i^1, a) = q_j^2$. We move to the state $q_j^2 \in M_2$.

Furthermore consider any transition $\delta(q_j^2, a) = q_k^2$ in M_2. We modify this transition as follows. We replace a by ϵ (to represent a skipped character) and we replace q_k^2 by q_k^1 so that we come back to the machine M_1. Thus, we remove this transition and add the transition $\delta(q_j^2, \epsilon) = q_k^1$.

We make this transformation on all the transitions of the machine M and get M'. The start state of the new machine is the start state of M_1. For the final states, we will have the original final states of M_1 and we will also have the original final states in M_2 (to capture the cases when y has odd length.)

To see that this new NFA accepts the language L_{alt}, note that for every string in $y_1y_2y_3 \ldots \in L$ there is an accepting path for the string $y_1\epsilon y_3\epsilon y_5 \ldots \in L_{alt}$. By similar arguments one can also see why any accepting string should have a corresponding string (when we fill in the skipped characters) in L, proving L_{alt} has a DFA and thus is regular.

4. **(30 points).** Let L be any language, let $L_{\frac{1}{2}-}$ be the set of all the first halves of strings in L so that

$$L_{\frac{1}{2}-} = \{x| \text{ for some } y, |y| = |x|, xy \in L\}$$

For any regular language L show that $L_{\frac{1}{2}\_}$ is regular. (*Hint: Think about the way we implemented two machines in "parallel" by using the Cartesian product. This idea maybe useful for this problem.*)

Solution. Denote n to be the length of x. Suppose at each step i, we keep track of the state q that M is in and also the set S_i of states that can reach a final state in exactly i characters. Then when the next input character c comes in, we advance the state of M to $\delta(q, c)$ accordingly, then find S_{i+1} by taking S_i and following transitions backwards. If input finishes after n steps, then we can accept if M's state appears in S_n!

Keeping track of the state of M is easy, but let us find a simple way to keep track of S_i. For this, we use M_R. However, every non-ε-move of M^R will be over *any* alphabet symbol (call this new machine $\hat{M}^R$). Then if we feed i characters into $\hat{M}^R$, then the set of possible states it could be in is precisely S_i! Now, we simply use a cross-product construction to simultaneously run M and $\hat{M}_R$.

Our machine N will have state set $Q \times 2^Q$ (here, we ignore q_0' from M^R). The initial state will be (q_0, F) since M start in q_0 and $\hat{M}^R$ starts in F. When we are in state (q, S) and we read in symbol c, we move to state $(\delta(q, c), \{p \mid \delta(p, c') \in S, c' \in \Sigma\})$. Now, our accepting states are $\{(q, S) \mid q \in S\}$.

Notice that if $x \in L_{\frac{1}{2}\_}$ then let y be its second half such that $xy \in L$. Then clearly, if q is the state that M ends up in after reading x, then q must also have a path of length $|y|$ to a final state. This means that after reading x, our machine N will be in some state (q, S) with $q \in S$. Thus, N indeed accepts x. Similar logic shows that N accepts only those strings in $L_{\frac{1}{2}\_}$.