Theory of Computation '24
Problem Set 2

Notations. Let $\Sigma = \{a, b\}$. For $w \in \Sigma^\star$, let $|w|$ denote the length of w. Let $\#_a(w)$ denote the number of as in w and let $\#_b(w)$ denote the number of bs in w.

1. **Definition 1** *Let Σ and Γ be two finite alphabets. A function $f : \Sigma^* \longrightarrow \Gamma^*$ is called a string homomorphism if for all $x, y \in \Sigma^*$, $f(x \cdot y) = f(x) \cdot f(y)$.*

   Prove that the class of regular languages is closed under homomorphisms. That is, prove that if $L \subseteq \Sigma^*$ is a regular language, then so is $f(L) = \{f(x) \mid x \in L\}$. Here, it is advisable to informally describe how you will turn a DFA for L into an NFA for $f(L)$.

   Solution.
   We make the following observation about the *homomorphism* operation. Given $f(L)$, for any $x = x_0 \cdot x_1 \cdots x_n \in L$, it can be proven recursively that

   $$f(x) = f(x_0 \cdot x_1 \cdots x_n) = f(x_0) \cdot f(x_1) \cdots f(x_n)$$

   Let's see how to convert a DFA for L to an NFA for $f(L)$ informally. An NFA for $f(L)$ can keep the *same* transitions from one state to the next. The catch is that each alphabet σ for a transition needs to be replaced by its image $f(\sigma)$, which is a string. Now, how do we go from one state p to another state q using some string $f(\sigma)$? We introduce artificial states in between p and q. Specifically, we add a DFA D recognizing the language $\{f(\sigma)\}$ in the middle and add epsilon-transitions from state p to the start state of D and from the accept state(s) of D to q. Note that the accept states of D are not the accept states of our NFA for $f(L)$. (The same can be done by using an NFA recognizing $\{f(\sigma)\}$).

   Formally, let $M(L) = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes a given regular language L. Let $M(f(\sigma))$ denote the DFA recognizing $\{f(\sigma)\}$ for each $\sigma \in \Sigma$. Then, we construct an NFA $N(f(L)) = (Q', \Gamma, \delta', q_0', F')$.

   - $Q' = \{q \mid q \in Q$ or q is a state in $M(f(\sigma))$ where $\sigma \in \Sigma\}$
   - Γ is the language of the homomorphism
   - $q_0' = q_0$
   - $F' = F$
   - The transition function $\delta' : (Q' \times \Gamma) \to \mathbf{2}^{Q'}$ will be loosely defined as follows. We will need multiple copies of each $M(f(\sigma))$. We replace the transition $\delta(q_1, \sigma) = q_2$ between two states $q_1, q_2 \in Q$ as follows. Take a copy C of $M(f(\sigma))$ and place it between q_1 and q_2. Draw out ϵ-transitions from q_1 to the start state in C and from accept states of C to q_2.

It is easy to realize that $M(f(L))$ certainly recognizes $f(L)$. To see this, consider any string $x = x_0 \cdot x_1 \cdots x_n \in L$. This string goes through some sequence of states in $M(L)$ as follows.

$$q_0 = q^{(0)} \xrightarrow{x_0} q^{(1)} \xrightarrow{x_1} q^{(2)} \cdots q^{(n)} \xrightarrow{x_n} q^{(n+1)}$$

where $q^{(i)}$ represents the state reached after reading $x_0 \cdot x_1 \cdots x_{i-1}$. By construction, $f(x) = f(x_0) \cdot f(x_1) \cdots f(x_n) \in f(L)$ goes through a *similar* sequence of states in $N(f(L))$.

$$q_0 = q^{(0)} \xrightarrow{f(x_0)} q^{(1)} \xrightarrow{f(x_1)} q^{(2)} \cdots q^{(n)} \xrightarrow{f(x_n)} q^{(n+1)}$$

The only difference is that reading each string $f(x_i)$ takes $N(f(L))$ through some additional intermediate states that help 'recognize' $f(x_i)$.

The final states of both the $M(L)$ and $N(f(L))$ are the same. So, if $q^{(n+1)} \in F$, then x is accepted in $M(L)$ and $f(x)$ in $M(f(L))$. Otherwise, both are rejected from their respective DFAs. In other words, we have proven that any string $f(x)$ is accepted in $N(f(L))$ if and only if x is accepted in $M(L)$. This means that $N(f(L))$ recognizes $f(L)$. □

2. Let $L_k = \{x \in \{0,1\}^* \mid |x| \geq k$ and the k'th character of x from the end is a $1\}$. Prove that every DFA that recognizes L_k has at least 2^k states. Also show that, on the other hand, there is an NFA with $O(k)$ states that recognizes L_k.

   Solution. This one can be proved directly using Pigeon Hole principle. But now that we have seen the notion of distinguishable strings and proved a lower bound on the number of states in the DFA using that, let us use it.

   Consider any two different k-bit strings $x = x_1 \cdots x_k$ and $y = y_1 \cdots y_k$ and let i be some position such that $x_i \neq y_i$ (there must be at least one such position). Hence, one of the strings contains a 1 in the ith position, while the other contains a 0. Let $z = 0^{i-1}$. Then z is a distinguishing extension of x and y with respect to L since exactly one of xz and yz has the kth bit from the end as 1. Since there are 2^k binary strings of length k, which are all mutually distinguishable by the above argument, any DFA for the language must have at least 2^k states.

   NFA with $k+1$ states. We construct an NFA with $Q = \{0, 1, 2, \cdots k\}$, with the names of the states corresponding to how many of the last k bits the NFA has seen. Define $\delta(0,0) = 0, \delta(0,1) = \{0,1\}$ and $\delta(i-1, 0/1) = i$ for $2 \leq i \leq k$. We set $q_0 = 0$ and $F = \{k\}$. The machine starts at state 0, on seeing a 1 it may guess that it is the kth bit from the end and proceed to state 1. Observe that the transitions ensure that the NFA reaches state k and accepts if and only if there are exactly $k-1$ bits following the one on which it moved from 0 to 1.

3. A *devilish NFA* is same as an NFA, except that we define the acceptance criterion of a devilish NFA as follows. We say that a devilish NFA N accepts x if and only if every run of N on x ends in an accepting state. Prove that a language is recognized by a devilish NFA if and only if the language is regular.

   Solution.
   We learn two interpretations of the regular NFA - *it proceeds magically by guessing the correct choice at each step*, or *it parallely computes every possible path* of the run of a string. The second interpretation says (in a way) that the NFA keeps track of all the possible "states" you can be in after reading some string.

Proof.
($\Longrightarrow$) Given a language L that is recognized by a devilish NFA $N_D(L)$. We show that L is regular by converting $N_D(L)$ to an equivalent DFA $M(L)$. Similar to how we convert a regular NFA to a DFA, we consider the states of $M(L)$ to be the power set of states of $N_D(L)$, wherein each state $Q' \subseteq \mathbf{2}^Q$ represents that the runs of $N_D(L)$ are in *all states of* Q' *at once*. The transitions are also defined in the same manner as in the NFA to DFA problem. However, in this case, there is only 1 accept state, being $\{q \mid q \in F\}$, i.e. the subset exactly equal to F - because this state signifies that all runs of a string ended in an accept state.

Hence, we have shown that every devilish NFA can be converted into an equivalent DFA. Hence, any language recognized by a devilish NFA is regular.

($\Longleftarrow$) Given that a language L is regular, we show that it is recognized by some devilish NFA. Looking carefully at the definition of the devilish NFA, it states, *"A devilish NFA is the same as an NFA but ..."*. We know that every DFA is an NFA. Also, one string can run in only one way on a DFA. Hence, if a string is accepted by a DFA, then it is actually accepted by "all" runs of the string on the DFA. Then, by definion, every DFA is a devilish NFA. Since L is regular, there exists a DFA $M(L)$ recognizing it. By the above argument, this DFA is a valid devilish NFA for L.

4. If A is any language, let $A_{\frac{1}{2}-}$ denote the set of all first halves of strings in A so that

$$A_{\frac{1}{2}-} = \{x \mid \text{for some } y,\ |x| = |y| \text{ and } xy \in A\}$$

Show that if A is regular, then so is $A_{\frac{1}{2}-}$.

Solution. Here is one way to do this: Lets say A is given by an NFA (we will abuse notation to call this NFA also A) $A = (Q, \Sigma, \delta, q_0, q_{acc})$. Here we use one fact that any NFA with multiple accept states can be simulated by an NFA with only a single accept state (why?).

Let's build an NFA $B = (Q', \Sigma, \delta', q_0', F')$.

Then define B as follows:

- The states Q' are of the form $[q_1, q_2]$ where $q_1 \in Q, q_2 \in Q$
- The start state of B is $[q_0, q_{acc}]$.
- $\delta'([q_1, q_2], a) = \{[u, v] \in Q \times Q : u \in \delta(q_1, a), \exists b \in \Sigma, q_2 \in \delta(v, b)\}$
- The accepting states of B are $F' = \{[q, q] : q \in Q\}$.

Intuition. The main idea behind the above construction is the following. The computation on NFA B can be thought of as 'simulating' two different computations on A using the 'pairs' - one forward, another one backward. Imagine putting a finger on the start state of A (which is q_0) and another one on the accept state of A. Now, suppose you are read one symbol from the input to B - this is possibly the first symbol of a potential x. which can be accepted. You move one finger in A as per the transition. However, the second finger does not really know where to go - why? Well because you do not know what y is, leave aside knowing it's end. But here in comes the power of non-determinism. You just 'guess' the last symbol of y and move to that state. This is easily simulated in B using the construction we have done.

Now it is not difficult to see that you will accept x if at the end of x, both your fingers are exactly at the same position in A. This is simulated by the accept states defined for B.

To prove formally, let x be a string accepted by the NFA B. Also, let $\hat{\delta'}$ be the extended definition of the transition function δ' such that $q' \in \hat{\delta'}(q, x)$ if and only if there exists a path in NFA B from q to q' labelled with the string x. We prove that then $\exists y$ such that $xy \in A$. By definition of accept states in B, $\hat{\delta'}([q_0, q_{acc}] = [q, q]$ for some $q \in Q$. We make two observations now. Each state in B is a 2-tuple. Now, consider the 2-tuples on the path denoted by $\hat{\delta'}([q_0, q_{acc}], x)$. By definition, if $\delta'([q_1, q_2], a) = [q_1', q_2'])$ for some $a \in x$, then $q_1' \in \delta(q_1, a)$. Hence $\hat{\delta}(q_0, x) = q$. On the other hand, there exists some symbol a' such that $q_2 \in \delta(q_2', a')$. Thus, there must exist some string y such that $q_{acc} \in \hat{\delta}(q, y)$. (Ideally we should prove the last two by induction but this is sort of an overkill here). Also, it is trivial to observe that $|x| = |y|$ Hence xy is accepted by A.

Conversely, consider a string $x \in A_{\frac{1}{2}^{-1}}$. Hence, there exists a y such that $|x| = |y|$ and $xy \in A$. We first prove the following by induction. For every $0 < \ell \leq |x|$, let where x_ℓ be the ℓ-length prefix of x. Further, let $\hat{\delta}(q_0, x_\ell = q_1'$ and there exists a string y_ℓ of length ℓ such that $q_{acc} \in \hat{\delta}(q_2', y_\ell)$. Then $\hat{\delta}(q_0, x_\ell) = [q_1'.q_2']$. I am leaving the proof if this hypothesis to you - its not hard.

Once we have this, let us now go back to the string $xy \in A$ where $|x| = |y|$. Hence, there has to be a state q such that $q \in \hat{\delta}(q_0, x)$ and $q_{acc} \in \hat{\delta}(q, y)$. Thus by the above proof, $\hat{\delta}(q_0, x_\ell) = [q, q] \in F'$. Thus x is accepted by B.

5. Write the regular expressions corresponding to the following languages.

   a. $L = \{\#_a(w) = 1(\mod 2)\}$.
   Solution. So this one wants you to find a reg-ex for strings that contain an odd number of a's. A nice and standard technique to solve these problems is to write down examples and break down the string in to patterns that are going to be repeated. In this case, since you need an odd number of a's (that is numbers of the form $2k + 1$, for $k \geq 0$), observe that any string with this property can be broken down as follows - begin with zero or more b's, followed by an a (this takes care of the $+1$). Now the following pattern repeats - zero or more b's followed by another a, again followed by zero or more b's and finally an a (this takes care of the $2k$ part). Finally there could be bunch of b's at the end. Hence, now it is easy to see that one possible reg-ex for L is

   $$(b^\star a)(b^\star a b^\star a)^\star b^\star$$

   Again, the solution itself is not so important. What you should remember is the technique. Also note that there could be simpler regular expressions for the above.

   b. $L = \{$every other letter in w is $a\}$ **Solution.** Very easy.

   c. $L = \{w$ contains an odd number of a's and an even number of b's$\}$ **Solution. Solution.** This is a tricky one. Here is one possible solution. Observe that any string with the above property can broken in to three possible parts : the first part contains an even number of a's and b's, the second part is one of the following - just a single a (this makes the number of a's odd) or a bab , followed by a third part which has zero or more occurrences of bb. Now the problem reduces to figuring out the first part, that is reg-ex for strings with even number of a's and b's. This is slightly simpler than the original task. Here is an observation, any such string can be thought of as zero or more repetition of one of the following patterns - $aa, bb, abba, baab, abab, baba$ (one can formally prove this but let us just be convinced for now). Now we can write the final reg-ex

$$(aa + bb + abba + baab + abab + baba)^\star(a + bab)(bb)^\star$$