Introduction to the Theory of Computation Homework #2 Solutions

(1. and 2. omitted)

3. (Exercise 1.13) Give regular expressions for all four languages in Exercise 1.4.
   Solution (there are multiple equivalent expressions in each case):

- $\{w \mid w \text{ begins with a 1 and ends with a 0}\} = 1(0+1)^*0$
- $\{w \mid w \text{ contains at least three 1s}\} = 0^*10^*10^*1(0+1)^*$
- $\{w \mid \text{every character at an odd position of } w \text{ is a 1}\} = (1(0+1))^*(1+\epsilon)$
- $\{w \mid w \text{ contains an even number of 0s, or exactly two 1s}\} = 1^*(01^*01^*)^* + 0^*10^*10^*$

4. (Exercise 1.10b) We saw in class that switching the accepting and non-accepting states of a DFA that recognizes the language L yields a DFA that recognizes the language $\overline{L}$. Show by example that this is not true for NFAs; that is, show an NFA where switching the accepting and non-accepting states does not complement the language it recognizes.

Solution: Here's a trivial example: consider an NFA M with $\Sigma = \{a\}$, start state q_0 and two more states q_1 and q_2. Let $\delta(q_0, a) = \{q_1, q_2\}$, i.e. M can move to either q_1 or q_2 upon reading a. If the accepting subset F is $\{q_1\}$, then M accepts the word a by moving to q_1; if we complement F then it still accepts a, this time by moving to q_2.

5. Let's formalize the idea of strings that are equivalent if they can be followed by the same suffixes. For a given language L, say $u \sim v$ if, for all w, $uw \in L$ if and only if $vw \in L$. This equivalence divides the set of strings Σ^* up into equivalence classes.

Suppose L is recognized by a DFA M. Now using the definition of δ^* we gave in class, prove formally that if $\delta^*(q_0, u) = \delta^*(q_0, v)$ — that is, if u and v lead to the same state of M from the start state — then $u \sim v$. (This proof only takes a few lines in our notation.)

Solution: If $\delta^*(q_0, u) = \delta^*(q_0, v)$, for any w we have $\delta^*(q_0, uw) = \delta^*(\delta^*(q_0, u), w) = \delta^*(\delta^*(q_0, v), w) = \delta^*(q_0, vw)$. Thus reading uw and vw puts the machine in the same state. If this is an accepting state, uw and vw are both accepted; if it is not, they are both rejected. Thus $uw \in L$ if and only if $vw \in L$, so $u \sim v$.

Conclude from this that the number of states of M must be at least as large as the number of equivalence classes defined by $\sim$.

Solution: Suppose $u \not\sim v$. (That is, there is at least one w such that $uw \in L$ but $vw \notin L$ or vice versa.) Then $\delta^*(q_0, u) \neq \delta(q_0, v)$, so they go to two different states. If there are k words, none of which are equivalent to each other (i.e. in k different equivalence classes) then there must be at least k different states for them to go to.

Now prove formally that the set of palindromes is not a regular language.

Solution: (I should have said "in any alphabet with at least two symbols," since any string in one-symbol alphabet is a palindrome!) Let $\Sigma = \{0, 1\}$. Then for any $i \neq j$, $0^i1 \not\sim 0^j1$ since $0^i10^i \in L$ but $0^j10^i \notin L$. Thus there are a countably infinite number of inequivalent states, so a DFA would need an infinite number of states (which is nonsense unless we give it some form of infinite memory, like a stack…)

6. (Exercise 1.17c plus a little more) Give two proofs, one based on the *pumping lemma* (which will will talk about this week) and one based on the previous exercise, that the language $\{a^{2^n} \mid n \in \mathbb{N}\}$ is not regular. Here $\Sigma = \{a\}$ and a^k means a string of k a's.

Solution (with the pumping lemma): The pumping lemma states that any sufficiently long word $s \in L$ can be written $s = xyz$ such that $xy^iz \in L$ for all $i \geq 0$, and moreover that $0 < |y| \leq p$ for some constant p. To show this leads to a contradiction, choose k so that $2^k > p$, and let $s = a^{2^k} \in L$ and write $s = xyz$. But $xy^2z = a^{2^k+|y|} \notin L$, since $2^k + |y| \leq 2^k + p < 2^{k+1}$ is less than the next largest power of 2 after 2^k. Thus the pumping lemma is false and the language is not regular.

Solution (with inequivalent words): I claim that $a^i \not\sim a^j$ for all $i \neq j$. To see this, assume without loss of generality that $i < j$, and choose k such that $2^k > j - i$. Now set $m = 2^k - i$. Then $a^ia^m = a^{2^k} \in L$. However, $a^ja^m \in L$ since $j + m = 2^k + j - i < 2^{k+1}$ is less than the next biggest power of 2 after 2^k. Thus there are an infinite number of inequivalent words and the language is not regular.

7. (Problem 1.31) Consider a new kind of automaton called an *All-Paths NFA*. This is just like an NFA, except we say it recognizes a word if *all* computation paths accept, unlike an NFA which accepts if at least one path does. (Note that what we have done logically is replace $\exists$, "there exists a path", with $\forall$, "for all paths".)

Show that All-Paths NFAs recognize exactly the regular languages, by describing how to convert an All-Paths NFA into an DFA that recognizes the same language.

Solution: We start just as in our conversion of an NFA to a DFA. Let the NFA have states Q with start state $q_0 \in Q$, accepting states $F \subseteq Q$ and transition function $\delta : Q \times \Sigma \to \mathcal{P}(Q)$. As before, we define a DFA with states $Q' = \mathcal{P}(Q)$, start state $\{q_0\}$ and transition function $\delta'(S, a) = \cup_{q \in S}\, \delta(q, a)$. However, whereas for a standard NFA we defined the DFA's accepting states as the subsets of Q that contain at least one accepting state, $F' = \{S \subseteq Q : S \cap F \neq \emptyset\}$, for the All-Paths NFA we define the DFA's accepting states as the subsets of Q whose elements are *all* accepting, i.e. they are contained in F. Thus $F' = \{S \subseteq Q : S \subseteq F\}$.

This proves one direction, that All-Paths NFAs recognize only regular languages. The other direction (that any regular language is accepted by some All-Paths NFA) follows from the fact that a DFA is an All-Paths NFA by definition.

8. (Problem 1.39) Prove that for all $k > 1$, DFAs with k states are more powerful than DFAs with $k - 1$ states. That is, come up with a family of languages L_k such that L_k can be recognized by a DFA with k states but not by a DFA with $k - 1$ states. Thus the DFAs form a strict hierarchy based on the number of states, which classify the regular languages according to their complexity.

Solution: for instance, let $\Sigma = \{a\}$ and let $L_k = \{a^i \mid i \geq k-1\}$ consist of words of length at least $k-1$. Clearly there are k inequivalent words, a^i for $0 \leq i < k$, since if $i \neq j$ then $a^i a^{k-1-i} \in L$ but $a^j a^{k-1-i} \notin L$. Thus no DFA with $k-1$ states (or fewer) can recognize L_k. On the other hand it is easy to construct a DFA with k states that does.

9. (Problem 1.44 — a little harder) Now prove that NFAs can be exponentially more compact than DFAs, or, to put it differently, that the exponential blowup in the number of states we get when converting an NFA to a DFA is sometimes necessary. That is, come up with a family of languages L_k such that L_k can be recognized by an NFA with k states, but that the smallest DFA that recognizes it has $\Theta(2^k)$ states. Hint: Look at Example 1.14 in Sipser.

Solution: Let L_k be the set of words in $\{0,1\}^*$ whose kth-to-last symbol is a 1. This is a generalization of Example 1.14, where $k = 3$. An NFA can recognize this language with $k+1$ states: the start state q_0, a state q_1 it can move to when it sees a 1, "guessing" that it's the kth-to-last symbol, and then a series of $k-1$ states which count the number of remaining symbols and end in the accepting state.

Now suppose that u and v differ somewhere in their last k symbols. Then without loss of generality there is some $1 \leq i \leq k$ such that the ith-to-last symbols of u and v are 1 and 0 respectively. Then $uw \in L$ but $vw \notin L$ for $|w| = k - i$, so $u \not\sim v$.

Since there are 2^k choices of the last k symbols, there are 2^k inequivalent words, so any DFA for L_k needs at least 2^k states. (In fact, 2^k states are sufficient as well as necessary, since a DFA with 2^k states can remember the most recent k symbols.)

10. (Problem 1.42 — Tricky!) For a language L, define $L_{1/2}$ as

$$L_{1/2} = \{x \mid xy \in L \text{ for some word } y \text{ such that } |y| = |x|\}$$

That is, $L_{1/2}$ is the set of "first halves" of L, namely the set of words x that can be followed by words y of the same length giving a word xy in L. Prove that if L is regular, so is $L_{1/2}$.

Solution: start with a DFA that accepts L with states Q and accepting states F. The idea is that for xy to end in an accepting state, x must end in a "modified accepting state," meaning a state which will end in an accepting state if we read y. Formally, we define the set of modified accepting states as $F_y = \{q \in Q : \delta^*(q, y) \in F\}$.

As we read x, we guess the symbols of y in reverse order. Specifically, when we read the ith symbol of x, we guess the ith-to-last symbol b of y. Let $y(i)$ denote the last i symbols of y; then we modify the accepting states from $F_{y(i-1)}$ to $F_{y(i)} = \{q \in Q : \delta(q, b) \in F_{y(i-1)}\}$. After we have read all of x we have guessed F_y.

To turn this into an NFA for $L_{1/2}$, we just need to expand the set of states from Q to $Q \times \mathcal{P}(Q)$ so we can keep track both of the DFA's state and the modified $F_y \in \mathcal{P}(Q)$. Then our accepting states are $\{(q, F_y) : q \in F_y\}$.

11. (Problem 1.41) As we will discuss in class, the language

$$L = \{w \mid w \text{ has an equal number of 0s and 1s}\}$$

is not regular. However, prove that the language

$$L = \{w \mid w \text{ has an equal number of 01s and 10s}\}$$

is regular.

Solution: A little reflection reveals that this language is simply the set of words that start and end with the same symbol in $\{0, 1\}$; the number of 01s (and 10s) is the number of blocks of the other symbol. Thus this language is

$$0(0+1)^*0 + 1(0+1)^*1$$

(which is regular since it can be described with a regular expression).