CS487 Problem Set 1 Model Solutions

Grading Standard. Each problem counts for 10 points. You can always argue with me, and be sure to do this before you get angry :).

10 perfect;
8 correct with small problem(s);
6 basically correct, serious problem(s);
4 bassically incorrect, but on the right way;
2 totally wrong, but make efforts;
0 blank.

Acknowledgement. Thanks to Michael Kazhdan and Spyridon Triantafyllis for providing their latex files.

Problem 2. Let R be the regular expression describing L. Then the expression $R' = \Sigma^* R \Sigma^*$ is precisely the regular expression which accepts all strings containing a word of L as a substring. Or in other words, R' describes the language $\overline{L'}$. But since $\overline{L'}$ is regular this implies that $L = \overline{\overline{L'}}$ is regular as well.

Problem 3. Solution 1: The idea of the proof is to take an automaton M describing the language L, create an automaton M' from M which accepts a word w if and only the reverse of w is accepted by M. We let M'' be the automaton M' with every transition labelled by some $\sigma \in \Sigma$ replaced by a transition labelled by Σ. Then given a word w we will run the M and M'' in parallel and accept w if after reading the word, the two machines end up in the same state. (For if the automaton M'' can get to state q after reading w this implies that there is some word of length $|w|$ which will take M from state q to a final state. If in addition the word w can get M from the start state to state q, this implies that w is in fact the first half of some word accepted by M.)

Let $M = (Q, \Sigma, \delta, q_0, F)$. We create a new state $q_0' \notin Q$ and define the automaton $M = (Q', \Sigma, \delta', q_0', \{q_0\})$ with:

$$\begin{array}{rcl} Q' & = & Q \cup q_0' \\ \delta(q,\sigma) & = & \begin{cases} F & \text{if } q = q_0' \text{ and } \sigma = \varepsilon \\ \{q' \in Q | \delta(q',\sigma) = q\} & \text{otherwise} \end{cases} \end{array}$$

$L(M')$ is the reversal of $L(M)$.

We now define the automaton $M'' = (Q', \Sigma, \delta'', q_0', \{q_0\})$ with:

$$\delta''(q,\sigma) = \begin{cases} \delta'(q,\varepsilon) & \text{if } \sigma = \varepsilon \\ \{q' \in Q | \exists \sigma' \in \Sigma \text{ s.t. } q' = \delta'(q,\sigma')\} & \text{otherwise} \end{cases}$$

M'' will accept a word w if there is some word of length $|w|$ such that M' accepts w.

Finally we define the automaton

$$M_{\frac{1}{2}-} = (Q_{\frac{1}{2}-}, \Sigma, \delta_{\frac{1}{2}-}, q_{\frac{1}{2}-}, F_{\frac{1}{2}-})$$

with:

$$\begin{aligned} Q_{\frac{1}{2}-} &= Q \times Q' \\ \delta_{\frac{1}{2}}((q, q'), \sigma) &= (\delta(q, \sigma), \delta''(q', \sigma)) \\ q_{\frac{1}{2}-} &= (q_0, q'_0) \\ F_{\frac{1}{2}-} &= \{(q, q) | q \in Q\} \end{aligned}$$

Then $L(M_{\frac{1}{2}-} = L_{\frac{1}{2}-}$ as desired.

Solution 2: Let $M = \langle Q, \Sigma, \delta, q_1, F\rangle$ be a DFA recognizing L, and let $Q = \{q_1, \dots, q_n\}$. We will construct an NFA N recognizing $L_{\frac{1}{2}-}$ using the following idea.

Given a string w, let us guess (non - deterministically) which state M will be in after parsing w. Let that state be q_i. We' ll build a sub-automaton which verifies that w leads M from q_1 to q_i, and that there is a path in M of length $|w|$ from state q_i to some final state. This sub-automaton will contain states of the form (i, q_i, q_j), where $q_i, q_j \in Q$. In this sub-automaton there will be a transition from state (i, q_j, q_k) to state $(i, q_{j'}, q_{k'})$ on input $a \in \Sigma$ iff:

- $\delta(q_j, a) = q_{j'}$ and
- $\exists b \in \Sigma$ such that $\delta(q_k, b) = q_{k'}$.

The "start state" of this automaton will be (i, q_1, q_i), and its final states will be $\{(i, q_i, q_k) \mid q_k \in F\}$. So, if this sub-automaton accepts a string w going through the states

$$(i, q_1, q_i), \dots, (i, q_j, q_k), \dots, (i, q_i, q_f)$$

then the sequence of states $q_1, \dots, q_j, \dots, q_i$ simulate a transition of the original DFA M from state q_1 to state q_i on input w, while the sequence of states $q_i, \dots, q_k, \dots, q_f$ simulates a possible transition of the original DFA M from state q_i to a final state, for some input of length $|w|$.

So, to build an NFA for the language $L_{\frac{1}{2}-}$ we need n copies of the above sub-automaton, one for each state $q_i \in Q$, and a start state q_0 which has only ϵ-transitions to all the states $(i, q_1, q_i), 1 \le i \le n$.

More formally, $N = \langle Q', \Sigma, \delta', q_0, F'\rangle$, where:

$$Q' = \{q_0\} \cup \{(i, q_j, q_k) \mid 1 \le i, j, k \le n\}$$

$$F' = \{(i, q_i, q_k) \mid 1 \le i \le n, q_k \in F\}$$

and δ' is defined by the following relations:

$\delta'(q_0, \epsilon) = \{(i, q_1, q_i) \mid 1 \le i \le n\}$
$\delta'(q_0, a) = \emptyset, \quad a \ne \epsilon$
$\delta'((i, q_j, q_k), a) = \{(i, q_{j'}, q_{k'}) \mid \delta(q_j, a) = q_{j'} \wedge (\exists b \in \Sigma)(\delta(q_k, b) = q_{k'})\}$

The previous analysis demonstrates that N recognizes the language $L_{\frac{1}{2}-}$. Thus, $L_{\frac{1}{2}-}$ is regular.

Problem 4.

1. A DFA is clearly an All-Paths-NFA, since for any input there is only one path of accepting/rejecting. Therefore any regular language can be accepted by an All-Paths-NFA.
2. If a language L is accepted by an *All-Paths-NFA* M, then $\bar{L}$ is accepted by the NFA M' which is M with accepting and non-accepting states switched, since by definition of accepting of an All-Paths-NFA, an input w is not in L iff there exists a path that leads to a non-accepting state. Therefore $\bar{L}$ is regular, which implies L is regular.

Problem 5. It suffices to show that $(L(M_1)\cap\overline{L(M_2)})\cup(\overline{L(M_1)}\cap L(M_2)) = \emptyset$. Therefore, the first step of our algorithm constructs a DFA N that accepts $(L(M_1) \cap \overline{L(M_2)}) \cup (\overline{L(M_1)} \cap L(M_2))$, and the second step decides if N accepts the empty language. The first step can be done by standar ways in the book, and the second by the elementary algorithm of reachability for the eaccepting states from the starting state.

Problem 6. It suffices to show that the equivalence relation on Σ^*:

$$a \approx b \text{ iff } (\forall c \in \Sigma^*)(ac \in L \leftrightarrow bc \in L)$$

defines an infinite number of equivalence classes.

Suppose that there is only a finite number of the above equivalence classes, and consider all the strings of the form 0^p, where p is a prime number. Since there is an infinite number of such strings, two of them, let's say 0^p and 0^q, have to be in the same equivalence class. This however is a contradiction, because $0^p1^q \in L$, whereas $0^q1^q \notin L$.

Problem 7.

1. $O(4^n)$. Let s_i be the maximum size of regular expressions labeled in the edges of the GNFA in our algorithm when i nodes have been deleted (call this GNFA M_i). Then $s_0 = 1$, and we want to bound s_{n-2}. Clearly $s_{i+1} \leq 4s_i$, since any regular expression in M_{i+1} can is $R_1^*R_2R_3^* \cup R_2$ for some R_1, R_2, R_3 in M_i.
2. $O(n)$ states and $O(n)$ transition edges. By induction on the length of the regular expression. Since in our construction algorithm, each adding one regular operator adds only constant number of states and transition edges.