CS–172 Computability & Complexity, Spring 2021

Homework 2 Solutions

Note: These solutions are not necessarily model answers. Rather, they are designed to be tutorial in nature, and sometimes contain more explanation than is required for full points. Also, bear in mind that there may be more than one correct solution. The maximum total number of points available is 29.

1. First, we observe that any DFA is an all-paths NFA, so all-paths NFAs accept all regular languages. For the other direction, we need to show that any language accepted by an all-paths NFA is regular. Let $N = (Q, \Sigma, \delta, q_0, F)$ be an arbitrary all-paths NFA. We will construct a standard NFA $M = (Q', \Sigma, \delta', q_0', F')$ that recognizes the same language as N. The construction will be similar to the conversion from NFAs to DFAs discussed in class (and in Theorem 1.19 of Sipser). The resulting NFA M will be almost a DFA, in that there will be at most one possible path in each computation. However, unlike in a DFA, every time we see a "dying" computation in N, the corresponding computation in M will die as well. (Alternatively, we could route such transitions to the dead state represented by the empty set, in which case we would end up with a DFA.) Note that this is where our construction differs from the NFA-to-DFA construction! *4pts*

$$
\begin{aligned}
Q' &= \mathcal{P}(Q) \\
q_0' &= \{q_0\} \\
\delta'(R, a) &= \begin{cases} \varnothing & \text{if for some } r \in R,\ \delta(r, a) = \varnothing; \\ \{q \in Q \mid q \in E(\delta(r, a)) \text{ for some } r \in R\} & \text{otherwise.} \end{cases}
\end{aligned}
$$

(Recall that $E(R)$ is the set of states reachable from R using zero or more ε transitions.) Finally, we want M to be accepting if its final state contains *only* accepting states of N, so we define $F' = \mathcal{P}(F)$.

An alternative argument is the following: First, augment the all-paths NFA N into an all-paths NFA N_1 such that N_1 accepts the same language as N, but no computation of N_1 dies. (One way to do this is to add a rejecting state q_{die} to N; then for all states $q \in Q \cup \{q_{\text{die}}\}$, add a transition from q to q_{die} labeled by all symbols in Σ unused by other outgoing arrows at q.) Now, reverse the accepting and rejecting states of N_1; call the resulting machine N_2. Think of N_2 as a standard NFA: The language accepted by N_2 is then the complement of the language L accepted by the all-paths NFA N_1. It follows that the complement of L is regular. Since regular languages are closed under complement (as we saw in lecture), L must be regular as well.

[*Note: In both possible solutions above, it is important to remember to handle the dying computations correctly. In the first solution, computations should not be allowed to proceed when one or more of their branches die; and in the second solution branches need to lead to the new state* q_{die}.]

2. Let $M = \{Q, \Sigma, \delta, q_0, F\}$ be a DFA for L. Informally, our NFA for $\frac{1}{2}(L)$ will work as follows: *4pts*

- First it "guesses" the state q in which M ends up on input x.
- It then verifies its guess by simulating M on x and making sure that the final state is indeed q.
- In parallel with the above, it simulates M on the "second-half" string y (of the same length as x), starting at state q and guessing each symbol of y as it goes.
- Finally, it accepts x iff the guess q is verified and the simulation on y ends in an accepting state of M.

We now describe how to implement the above idea. Let $M \times M$ denote the "product" DFA of M with itself, i.e., the state set of $M \times M$ is $Q \times Q$ and the transition from state $[q_1, q_2]$ on symbol a goes to $[\delta(q_1, a), \delta(q_2, a)]$. To guess the state in which M ends up on input x, the NFA needs $|Q|$ copies of $M \times M$, one for each potential guess, and an additional start state q_0' to get things going. From q_0' it has ε-transitions to the start state of each of these copies, which we designate $[q_0, q]^q$ to represent the fact that it is the "start" state of the $M \times M$ submachine particular to the guess q. Within this particular submachine, the NFA uses a trick similar to the product construction we saw in class in order to simulate the behavior of M on x starting from q_0 in parallel with its behavior on a guessed string y starting from q. For instance, let r, s be any two states of M, and suppose that M has transitions $\delta(r, a) = r_a$ and $\delta(s, b) = s_b$ for each $b \in \Sigma$. Then the new machine has the (nondeterministic) transition $\delta'([r, s]^q, a) = \{[r_a, s_b]^q : b \in \Sigma\}$, for every q. (Note that the multiple transitions from this state correspond to guessing the next symbol, b, of y.) Finally, the accepting states of the new machine are those of the form $[q, s]^q$ with $s \in F$: this ensures that (a) its guess q was correct, in that M did indeed end up in state q on x; and (b) the guessed string y (equal in length to x) takes M from state q to an accepting state. This means that there exists a y such that xy takes M from q_0 to an accepting state, which is exactly the condition that x belongs to $\frac{1}{2}(L)$.

To make the above precise, we can define the machine $\frac{1}{2}(M) = \{Q', \Sigma, \delta', q_0', F'\}$ as follows:

$$\begin{aligned} Q' &= [Q \times Q]^Q \cup \{q_0'\} \\ F' &= \{[q, s]^q : q \in Q, s \in F\} \end{aligned}$$

and transition function

$$\begin{aligned} \delta'(q_0', \varepsilon) &= \{[q_0, q]^q : q \in Q\} \\ \delta'([r, s]^q, a) &= \{[r', s']^q : \delta(r, a) = r' \text{ and } \exists b \in \Sigma \text{ with } \delta(s, b) = s'\} \quad \forall a \in \Sigma, \forall q, r, s \in Q. \end{aligned}$$

An alternative solution found by many students also uses the product construction but in a slightly different way. The idea is to again use pairs of states $[q, q']$, where now q records the current state of M as it reads x and q' records the current state of M as it reads the (guessed) second half string y *backwards*. To start, we use an ε-transition to go to a state $[q_0, q_f]$, where q_f is any accepting state of M. (This corresponds to guessing the accepting state reached after reading the full string xy.) Upon reading symbol a in state $[q, q']$, the machine transitions to any state of the form $[\delta(q, a), q'']$, where q'' ranges over all states that have a transition into q' (on any symbol): this corresponds to guessing the previous symbol of y. The machine accepts if and only if it ends up in a state of the form $[q, q]$ (meaning that the two computations, on x and on y, can indeed be glued together to form a valid computation on input xy, which is accepting because it ends up in a state q_f.

[*Most people attempted to use the product construction, as indicated in the hint. However, some people had trouble with the details. We graded this problem somewhat leniently, focusing on the main ideas rather than the details. Please carefully check the details of your construction and those above.*]

3. For each of these, there are many possible valid regular expressions. We omit the concatenation operator for clarity of notation.

(a) $\big(\big(\mathtt{a} \cup \mathtt{e} \cup \mathtt{i} \cup \mathtt{o} \cup \mathtt{u}\big)\Sigma^* \cup \varepsilon\big)\big(\mathtt{ing}\big)$ *2pts*

[*Some students forgot that the string* 'ing' *belongs to this language.*]

(b) $\mathtt{0}^*\big(\mathtt{10}^*\mathtt{10}^*\mathtt{10}^*\big)^*$ *2pts*

[*Important to remember to include strings with no 1's, and to allow sequences of 0's between each set of three 1's.*]

(c) $(\mathtt{10} \cup \mathtt{01})^*(\mathtt{1} \cup \mathtt{0} \cup \varepsilon)$ *2pts*

To see the above, first notice that every even length string that belongs in the language is such that *every* even-length prefix of it has equally many zeros and ones, because if not, then there are either two more zeros than ones or two more ones than zeros. It is easy to check, by induction on the length of the string, that the set of all even length strings where every even-length prefix has the same number of zeros and ones is given by $(\mathtt{10} \cup \mathtt{01})^*$. To then get our language, we simply concatenate either a zero or a one or nothing to the end of every even length string with the above property.

[*Some incorrect answers were* $(\mathtt{01} \cup \mathtt{10})^*(\mathtt{1} \cup \mathtt{0})$ *(which leaves out even-length strings), and* $(\mathtt{01} \cup \mathtt{10})^*$ *(leaves out odd-length strings).*]

4. (a) False. E.g., take $R = \mathtt{0}$ and $S = \mathtt{1}$. Then the string $\mathtt{010}$ belongs to $(R \cup S)^*$ but not to $R^* \cup S^*$. *2pts*

[*For False answers, you really have to give a counterexample; other attempted arguments are very unlikely to be convincing. Other counterexamples are possible.*]

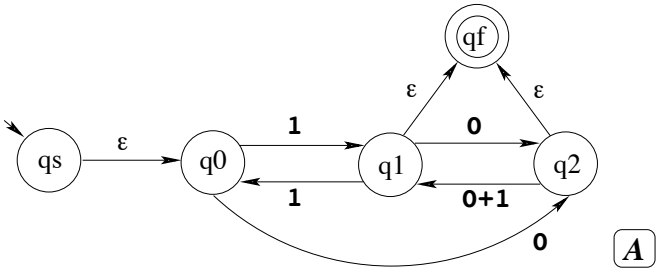
(b) True. The fact that $L(R^*) \subseteq L\big((R^*)^*\big)$ is immediate because the language on the right contains all words that consist of finite sequences of words from $L(R^*)$, so in particular it contains all words in $L(R^*)$. We also have to show that $L\big((R^*)^*\big) \subseteq L\,(R^*)$. To see this, note that any word in $L\big((R^*)^*\big)$ can be written in the form $w_1 w_2 \ldots w_n$ for some $n \geq 0$, where each w_i is a word in $L(R^*)$. But each w_i can in turn be written in the form $x_{i1} x_{i2} \ldots x_{im_i}$ for some $m_i \geq 0$, where each x_{ij} is a word in $L(R)$. So any word in $L\big((R^*)^*\big)$ can be written as a sequence of words from $L(R)$, and hence belongs to $L(R^*)$. Thus $L\big((R^*)^*\big) \subseteq L\,(R^*)$, as claimed. *3pts*

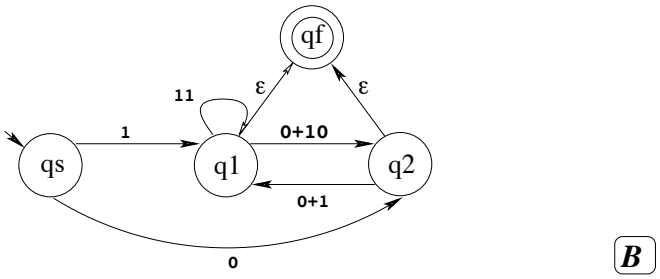
[*Arguments based on conversion to NFAs etc. are unlikely to work. The only really convincing way to do this is to prove set containment in both directions.*]

(c) False. E.g., take $R = \mathtt{0}$ and $S = \mathtt{0} \cup \varepsilon$. Then $L(R^*) = L(S^*) = \{0\}^*$, but $L(R) = \{0\}$ and $L(S) = \{0, \varepsilon\}$. *2pts*

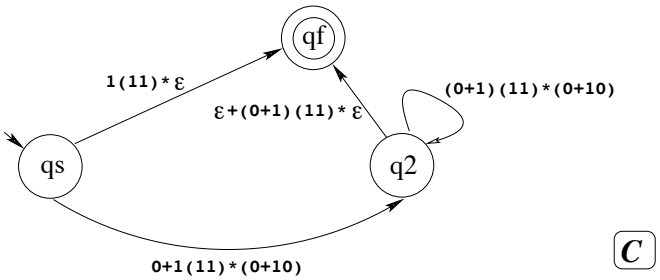
[*As in part (a), this needs a counterexample, and other counterexamples are possible.*]

5. See the attached figures. We follow the construction in class and in Sipser. The starting GNFA is shown in diagram A, with ε-transitions out of the start state q_s and from the states q_1, q_2 into the accepting state q_f. All missing arrows (i.e., from q_s to all states except q_0, from q_0 to q_f, from q_2 to q_0 and self-loops on q_0, q_1, q_2) are labeled with $\emptyset$. State q_0 is removed in diagram B, followed by q_1 and q_2 in diagrams C and D. The result is the regular expression $(\mathtt{0} \cup \mathtt{1}(\mathtt{11})^*(\mathtt{0} \cup \mathtt{10}))((\mathtt{0} \cup \mathtt{1})(\mathtt{11})^*(\mathtt{0} \cup \mathtt{10}))^*(\varepsilon \cup (\mathtt{0} \cup \mathtt{1})(\mathtt{11})^*) \cup \mathtt{1}(\mathtt{11})^*$. In the diagrams, for typographical reasons the union operator $\cup$ is shown as $+$. By noting that $(\mathtt{0} \cup \mathtt{1}(\mathtt{11})^*(\mathtt{0} \cup \mathtt{10}))$ is equal to $\mathtt{1}^*\mathtt{0}$ and that $(\mathtt{11})^*(\mathtt{0} \cup \mathtt{10})$ is equal to $\mathtt{1}^*\mathtt{0}$ we see that the above regular expression can be simplified to $\mathtt{1}^*\mathtt{0}((\mathtt{1} \cup \mathtt{0})\mathtt{1}^*\mathtt{0})^*(\varepsilon \cup (\mathtt{1} \cup \mathtt{0})(\mathtt{11})^*) \cup \mathtt{1}(\mathtt{11})^*$ (other simplifications are possible). *3pts*

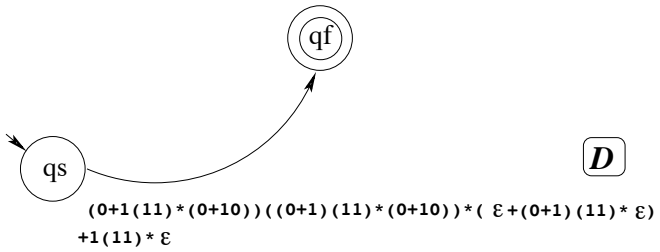




B



C



D

[*Some people made minor errors, especially when removing state q_1. We graded this problem quite leniently and did not penalize most minor errors. However, you should carefully compare your solution to the one above and check the details.*]

6. (a) `> egrep '[0-9]' data.txt` *5pts*
[**OR** `> egrep '[0123456789]' data.txt`]

```
In the summer of the year 1797, the Author, then in
Samuel Taylor Coleridge, 1772-1834
```

(b) `> egrep 'dome(.*)pleasure|pleasure(.*)dome' data.txt`

```
A stately pleasure-dome decree :
     The shadow of the dome of pleasure
A sunny pleasure-dome with caves of ice !
```

(c) `> egrep '.*(and ).*(and )' data.txt`

```
          pen, ink, and paper, instantly and eagerly wrote down the
          Porlock, and detained by him above an hour, and on his return
               Vanishes, and a thousand circlets spread,
```

(d) `> egrep '[,\.\!]' data.txt`
The (rather long) output is omitted here.

(e) `> egrep -v '[,\.!]' data.txt`
The output here consists of all lines not output in part (d).