CSE396 Problem Set 2 Answer Key Spring 2017

(1) For each of the following regular expressions r, design both an NFA N such that $L(N) = L(r)$ and a DFA M such that $L(M) = L(r)$. You are not expected to convert N into M by the text's procedure (which won't be covered until Tuesday) but rather to design N and M by strategy—and in particular, your M needs to have explanatory comments on its states (even if you do use the conversion). You can skip comments on the NFA—making it "resemble" r for direct readability is a point of the problem—but for full credit, your NFA should have fewer instructions than *any DFA you could possibly think to build.* This implies that you should try to economize by not using so many (or any) ϵ-arcs compared to the general-and-abstract proof of the r-to-N conversion in the text and lecture. All the expressions and languages are over $\Sigma = \{a, b\}$. $(3 \times (6 + 9) = 45$ pts.)

- $r = (ab)^*(ba)^*$.
- $r = (a \cup b)^*(aba \cup bba)$.
- $r = (aba \cup b)a^*$. *Answers in words:*

(a) The NFA has accepting states s, f with an ϵ-arc from the start state s to f. It then has instructions $(s, a, p), (p, b, s)$ carrying out the initial $(ab)^*$ loop, and $(f, b, q), (q, a, f)$ doing the other loop, where p and q bring the numbers of states to 4 to go with the 5 instructions.

The DFA has these 4 states but instead of (s, ϵ, f) it has (s, b, q), with the comment that an initial b (or any b after a bunch of $ababab\ldots$ repeats) commits you to being in the middle of the "ba" cycle. Thus far it is tied with the NFA in the number of instructions, but you need to add 3 more going to a dead state and 2 more there to call it a DFA, for a total of 10.

(b) The NFA has self-loops on both a and b at the state state with instructions $(s, a, s), (s, b, s)$. Then it has arcs on a and b going to a new state q. There are two more states r, f with instructions (q, b, r) and (r, a, f), making a total of 4 states and 6 instructions. No ϵ-arcs were needed.

The DFA has the idea that the regular expression is really $(a\cup b)^*(a\cup b)ba = (a\cup b)^+ba = (a \cup b)(a \cup b)^*ba$. The benefit of re-factoring it this way is now we see the idea of just having $(s, a, q), (s, b, q)$ as the options at the start state and allowing you to loop at q if needed. Now the game at q becomes "strings that end in ba." Run your instructions (q, b, r) and (r, a, f) out like class showed with other examples, but don't make f a "nirvana" state because this is "ends-with ba" not "has-a ba." So getting another char in state f knocks you back either to r (on a b) or all the way back to q (on an a). And getting another b in the midpoint state r preserves your progress and keeps you there; likewise an a at state q makes no progress but doesn't kill you. So in fact there's no dead state. But you do need 8 instructions, the other 4 being (q, a, q), (r, b, r), (f, a, q), and (f, b, r), which are 2 more than the NFA has.

(c) The NFA has one edge (s, b, f) going right to the final state f and one "detour" $(s, a, p), (p, b, q), (q, a, f)$, then finally a loop (f, a, f). This design actually has no nondeterminism—nor ϵ arcs. So to get the DFA, just add a dead state and route the "missing" arcs to it. Unlike (a), this DFA isn't even "morally different" from the NFA—but it still counts as having more instructions.

(2) (a) Describe in precise words a procedure by which you can take any NFA $N = (Q, \Sigma, \delta, s, F)$ and produce an NFA $N' = (Q', \Sigma, \delta', s', F')$ such that $L(N') = L(N)$ but F' consists of a single state f. That is to say, without loss of generality, we could have defined NFAs in the text to have only one accepting state from the get-go.

(b) Now revisit your answer and try to do it *without adding any extra ϵ-arcs.* You may make one assumption—if you need to—which is that $\epsilon \notin L(N)$. Can you do it now? (12 + 12 = 24 pts., for 69 on the set)

Answer: (a) Given any NFA $N = (Q, \Sigma, \delta, s, F)$, add a new state f and arcs (q, ϵ, f) from every state $q \in F$. The re-define F to be $F' =$ just $\{f\}$. The resulting NFA N' has just the one accepting state and makes $L(N') = L(N)$ because: if x is processed by N from s to a final state q then we can take one more null step to complete processing the same $x \cdot \epsilon = x$ from s to f. So $L(N) \subseteq L(N')$. And conversely, if $x \in L(N')$, then the only way we could have processed x from s to f in N' was to have gone through some final state q of N before the final ϵ-jump. So N can process x from s to q, so $x \in L(N)$, so $L(N') \subseteq L(N)$ since x is general, so $L(N') = L(N)$ by both halves of the proof.

OK, the proof wasn't needed but it's good to read, also in 20 years we may have credential-entry super-CAPTCHA protocols that are more like proofs. And if you want to be really wonky, you could write $N' = (Q', \Sigma, \delta', s', F')$ where $Q' = Q \cup \{f\}$, $s' = s$, $F' = \{f\}$, and—drumroll please—

$$\delta' = \delta \cup \{(q, \epsilon, f) : q \in F\}.$$

(b) Now the idea is that we want to "look ahead" by 1 char: If c lets you go from some state p to some state $q \in F$, then you camn add an extra option to go to the new f again. That is, add f and add (p, c, f) for every state p and char c such that N can process c to a state in F. Now there's a little subtlety: An instruction (p, c, q) might not have $q \in F$ directly—instead it might be that you get from q to a state $r \in F$ via ϵ-arcs. This is another example where my "can process" lingo is useful. Then $L(N') = L(N)$ because $L(N') \subseteq L(N)$ is like before, and $L(N) \subseteq L(N')$ because for every $x \in L(N)$ with final character c, N' can process that final char directly to f instead.

Ah but—this statement needs there to BE a final char c in x. If $x = \epsilon$ and $\epsilon \in L(N)$ this logic doesn't hold deuterium oxide. In fact, without the restriction $\epsilon \notin L(N)$, the whole thing can become impossible. Here is the simplest counterexample I know: Take $L = \{\, x : \#1(x) \neq 1 \pmod 3 \,\}$. (It doesn't matter if the alphabet is $\{\, 0, 1 \,\}$ or just $\{\, 1 \,\}$.) Then $\epsilon \in L$ and we can make an easy DFA for L with 3 states that go in a cycle on 1s, which "Is-A" NFA N. Now we're not allowed to add any ϵ-arcs to N, so we must keep s as an accepting state. We can try to re-route and re-design the rest of the machine, even changing some old arcs, but we're killed by this reasoning if we try to keep s as the only accepting state: All options from s on 1 have to go to rejecting states since $1 \notin L$. But $11 \in L$ so you'd have to bring 11 back to s. But then 111 gives you only the previous rejecting states as options, which breaks the fact that $111 \in L$. So it can't be done and hence the extra clause really was needed for the original proof.