Discussion

It always pleases me when students come up with elegant solutions and insights that had not occurred to me when I created the exam. I experienced this pleasure several times while grading your work. Here are the most notable examples.

- Several students solved #5 without having to write out an example string in either $(L_1^* L_2^*)^*$ or $(L_1 \cup L_2)^*$ but instead working purely with sets. Marco Adelfio, Justin Bush and Paul Rosania had especially elegant solutions of this form.
- For #6.4, Marco Adelfio and Ben Minkowsky made the brilliant observation that $\textsc{Select}(L) = \overline{\textsc{Permute}(\overline{L})}$, which leads to a very clean solution based on the solution to #6.3. I like this a lot better than my "direct" solution which appears below, which uses much the same idea (of complements) but hides this cool observation.
- Almost everyone who solved #7 correctly did so by constructing a CFG for the language and then turning it into a PDA using the standard algorithm. The resulting solution is easier to understand than my direct PDA construction below.
- Anne Loomis produced a PDA directly for #7 and came up with a lovely 4-state solution.
- Ben Minkowsky was the only one to solve #4 using a regular expression approach, rather than the automaton approach I describe below. Ben's idea is that every regular language L has a regular expression which can be systematically "reversed" by following some simple rules to get a regular expression for $L^{\mathcal{R}}$; the correctness of these rules can then be proven by induction.

Unfortunately, there are also a few negative observations to be made. If any of the following describes your work, I urge you to make sure you understand why you went wrong and what you should have done instead. Read my solutions below and, if you like, ask your classmates to explain theirs to you. The more you talk about the material in the course, the better your understanding will be.

- Many of you will see the comment "NRE" next to some of your answers. This stands for "Not Rigorous Enough."
- A small number of solutions lost a little credit thanks to being overly verbose or overly convoluted. While you are not required to come up with ultra-elegant solutions, you *are* required to think a little about simplification.
- It appears that a lot of you were stumped by #6.3 and #6.4 in the sense that you didn't quite know what you were expected to do. The statements in these two problems are both false, so if you tried to prove that they were true then of course you were bound to fail. If you did suspect that they were false, you should have realized that a proof of #6.3 (say) would require coming up with a language L such that $\textsc{Permute}(L)$ is not regular. To prove the "not regular" part of this statement you clearly had to use the pumping lemma, as that is the only technique we have studied for proving non-regularity. However, very few of you even wrote the magic words "pumping lemma" in your solution to #6.3.
- Please note that you don't get credit for a correct but justificationless answer on TRUE/FALSE questions.

The Solutions

1. Write a regular expression for the language generated by the following grammar:

$$\begin{aligned} S &\longrightarrow AT \\ T &\longrightarrow AAT \mid BBT \mid AA \\ A &\longrightarrow 0 \\ B &\longrightarrow 1 \end{aligned}$$

A single line answer will do; you don't have to justify or show any steps. Your regular expression should be as simple as possible.

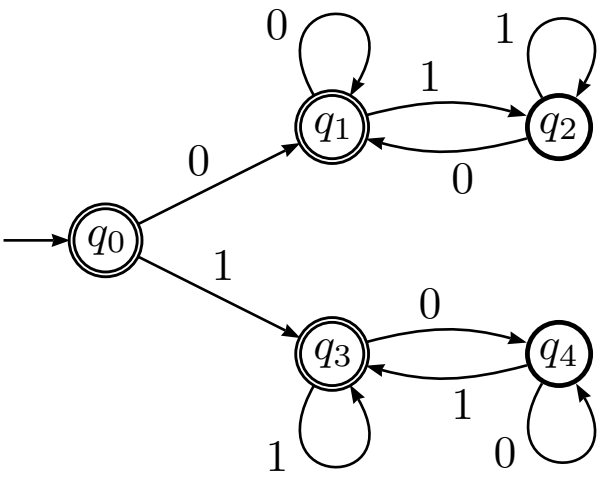
Solution: The grammar generates $0(00 \cup 11)^*00$.

2. Draw a DFA for the language

$$\{x \in \{0,1\}^* : x \text{ contains an equal number of occurrences of the substrings } 01 \text{ and } 10\}.$$

For example, 101 and 0000 are in the language, but 1010 is not.

Solution: The idea is to handle strings beginning with a 1 and strings beginning with a 0 separately. The following DFA does the job:



3. Suppose $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ are two DFAs over the same alphabet Σ. Consider the DFA $M = (Q_1 \times Q_2, \Sigma, \delta, (q_1, q_2), Q_1 \times Q_2 - F_1 \times F_2)$, where δ is given by

$$\forall q \in Q_1, q' \in Q_2, a \in \Sigma : \quad \delta((q, q'), a) \;=\; (\delta_1(q, a), \delta_2(q', a)).$$

3.1. What can you say about $\mathcal{L}(M)$, the language recognized by M? State your answer as a mathematical expression and keep it as simple as possible.
Solution: $\mathcal{L}(M) = \Sigma^* - \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$.

3.2. Prove that your answer above is correct.
Solution: First we prove that $\mathcal{L}(M) \subseteq \Sigma^* - \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$.

Suppose $x = a_1 a_2 \ldots a_n \in \mathcal{L}(M)$, where each $a_i \in \Sigma$. Then there exists a sequence of states $(r_0, s_0), (r_1, s_1) \ldots, (r_n, s_n)$ of states of M such that

i. $(r_0, s_0) = (q_1, q_2)$,
ii. $(r_i, s_i) = \delta((r_{i-1}, s_{i-1}), a_i) = (\delta_1(r_{i-1}, a_i), \delta_2(s_{i-1}, a_i))$, for $1 \leq i \leq n$, and
iii. $(r_n, s_n) \in Q_1 \times Q_2 - F_1 \times F_2$.

The properties of the sequence $r_0, r_1, \ldots, r_n$ imply that $\hat{\delta}_1(q_1, x) = r_n$. Likewise, the properties of the sequence $s_0, s_1, \ldots, s_n$ imply that $\hat{\delta}_2(q_2, x) = s_n$. Since $(r_n, s_n) \notin F_1 \times F_2$, either $r_n \notin F_1$ or $s_n \notin F_2$. In the former case, $x \notin \mathcal{L}(M_1)$, and in the latter case, $x \notin \mathcal{L}(M_2)$. Therefore, $x \in (\Sigma^* - \mathcal{L}(M_1)) \cup (\Sigma^* - \mathcal{L}(M_2)) = \Sigma^* - \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$.

To prove the other direction, $\Sigma^* - \mathcal{L}(M_1) \cap \mathcal{L}(M_2) \subseteq \mathcal{L}(M)$, we simply reverse the steps of the above reasoning (in brief, pick an $x \in \Sigma^* - \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$, construct appropriate sequences $r_0, \ldots, r_n$ and $s_0, \ldots, s_n$ from the two machines and then zip them together into a sequence $(r_0, s_0), \ldots, (r_n, s_n)$ for the machine M). We omit the details.

4. Recall that $x^{\mathcal{R}}$ denotes the reverse of the string x. For a language L, let $L^{\mathcal{R}} = \{x^{\mathcal{R}} : x \in L\}$. Prove that if L is regular, so is $L^{\mathcal{R}}$.

Solution: The idea is to start with a DFA for the regular language L and "reverse all the arrows" in this DFA, make its start state an accept state of the resulting machine, and make all its accept states start states (or rather, since only one start state is allowed, to introduce ε-transitions from a new start state to all the former accept states). This procedure clearly gives us an NFA, because, for instance, if there are five different states of the initial DFA that all lead into a state q on reading input symbol a, then after the conversion, q will lead to these five different states on input a. Now we give a formal proof.

Suppose L is a regular language. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA recognizing L. Let us construct an NFA $M' = (Q \cup \{q_{\text{new}}\}, \Sigma, \delta', q_{\text{new}}, \{q_0\})$ where δ' is given by

$$\begin{aligned}
\delta'(q_{\text{new}}, \varepsilon) &= F\,, \\
\delta'(q_{\text{new}}, a) &= \emptyset\,, && \forall\, a \in \Sigma\,, \\
\delta'(q, \varepsilon) &= \emptyset\,, && \forall\, q \in Q\,, \\
\delta'(q, a) &= \{r \in Q : \delta(r, a) = q\}\,, && \forall\, q \in Q,\, a \in \Sigma\,.
\end{aligned}$$

We claim that $\mathcal{L}(M') = L^{\mathcal{R}}$, which would prove that $L^{\mathcal{R}}$ is regular. To prove our claim, we need to argue that (1) $\mathcal{L}(M') \subseteq L^{\mathcal{R}}$ and that (2) $L^{\mathcal{R}} \subseteq \mathcal{L}(M')$.

To prove (1), consider an $x \in \mathcal{L}(M')$. By definition, this means that we can write $x = a_1 a_2 \ldots a_n$ with each $a_i \in \Sigma_\varepsilon$ and find a sequence $r_0, r_1, \ldots, r_n$ of states of M' such that

- $r_0 = q_{\text{new}}$, the start state of M',
- $r_i \in \delta'(r_{i-1}, a_i)$, for $1 \leq i \leq n$, and
- $r_n \in \{q_0\}$, the set of final states of M'.

By construction, a_1 must be ε, because otherwise the set $\delta'(r_0, a_1) = \delta'(q_{\text{new}}, a_1)$ would be empty. Also, the states r_{i-1} for $2 \leq i \leq n$ must all be different from q_{new}, and so, every a_i for $2 \leq i \leq n$ must be different from ε (i.e., in the set Σ). Therefore, by construction, $\delta'(r_{i-1}, a_i)$ for $2 \leq i \leq n$ is the set of all r such that $\delta(r, a) = r_{i-1}$. Since, by the second bullet above, r_i is in this set, it follows that $\delta(r_i, a_i) = r_{i-1}$. Finally, the state r_1 must lie in F, because it must lie in $\delta'(r_0, a_1) = \delta'(q_{\text{new}}, \varepsilon) = F$. Thus, we have

- $r_n = q_0$, the start state of M,
- $r_{i-1} = \delta(r_i, a_i)$, for $n \geq i \geq 2$, and
- $r_1 \in F$, the set of final states of M.

Thus we have a sequence $r_n, r_{n-1}, \ldots, r_1$ of states of M which satisfies the above three bullets; this ensures that $a_n a_{n-1} \ldots a_2 \in \mathcal{L}(M) = L$, i.e., that $x^{\mathcal{R}} \in L$, i.e., that $x \in L^{\mathcal{R}}$.

The proof of (2) is very similar, so we only sketch it here. We start with an $x \in L^{\mathcal{R}}$. This means $x^{\mathcal{R}} \in L = \mathcal{L}(M)$. Therefore $x^{\mathcal{R}}$ takes M through a sequence $r_0, r_1, \ldots, r_n$ of steps with $r_0 = q_0$ and $r_n \in F$. Arguing just as above, we can show that x can take M' through the sequence of states $q_{\text{new}}, r_n, r_{n-1}, \ldots, r_0$ and since q_{new} is the start state of M' and r_0 is an accept state of M', we see that M' accepts x. So $x \in \mathcal{L}(M')$.

5. Let L_1, L_2 be two languages over the same alphabet Σ. Prove that $(L_1^* L_2^*)^* = (L_1 \cup L_2)^*$. Remember that to prove $A = B$ for sets A and B you must separately prove the two statements $A \subseteq B$ and $B \subseteq A$.

Solution: Suppose $x \in (L_1^* L_2^*)^*$. By definition of Kleene star, x is a concatenation of zero or more strings, each in $L_1^* L_2^*$, i.e. $x = x_1 x_2 \ldots x_n$ with each $x_i \in L_1^* L_2^*$. Again, by definition, each $x_i = y_{i1} y_{i2} \ldots y_{is_i} z_{i1} z_{i2} \ldots z_{it_i}$ with each $y_{ij} \in L_1$ and each $z_{ij} \in L_2$. Putting it all together:

$$x = y_{11} \ldots y_{1s_1} z_{11} \ldots z_{1t_1} y_{21} \ldots y_{2s_2} z_{21} \ldots z_{2t_2} \ldots\ldots\ldots z_{nt_n} .$$

Since each y_{ij} and each z_{ij} is in $L_1 \cup L_2$, it follows that $x \in (L_1 \cup L_2)^*$. We have shown that $(L_1^* L_2^*)^* \subseteq (L_1 \cup L_2)^*$.

Now, suppose $x \in (L_1 \cup L_2)^*$. Then $x = x_1 x_2 \ldots x_n$ where each $x_i \in L_1 \cup L_2$. If an $x_i \in L_1$, we can write $x_i = x_i \varepsilon$ which puts it in $L_1^* L_2^*$. If an $x_i \in L_2$, we can similarly write $x_i = \varepsilon x_i$ which puts it in $L_1^* L_2^*$. Therefore, each $x_i \in L_1^* L_2^*$, whence $x \in (L_1^* L_2^*)^*$. We have shown that $(L_1 \cup L_2)^* \subseteq (L_1^* L_2^*)^*$.

This completes the proof.

6. A permutation of a string x is any string that can be obtained by rearranging the characters of x. Thus, for example, the string abc has exactly six permutations:

$$abc,\ acb,\ bac,\ bca,\ cab,\ cba .$$

Clearly, if y is a permutation of x, then $|y| = |x|$. For a language L over alphabet Σ, define

$$\begin{aligned} \textsc{Permute}(L) &= \{x \in \Sigma^* : x \text{ is a permutation of some string in } L\} , \\ \textsc{Select}(L) &= \{x \in \Sigma^* : \text{every permutation of } x \text{ is in } L\} . \end{aligned}$$

Classify each of the following statements as TRUE or FALSE, and give proofs justifying your classifications.

6.1. If $L_1 = 1^*0$, then $\textsc{Permute}(L_1)$ is regular.
Solution: TRUE. In this case $\textsc{Permute}(L_1) = \{x \in \{0,1\}^* : x \text{ contains exactly one } 0\} = 1^*01^*$, which is clearly regular.

6.2. If $L_1 = 1^*0$, then $\textsc{Select}(L_1)$ is regular.
Solution: TRUE. Any string in $\textsc{Select}(L_1)$ must clearly be in L_1 (since a string is always a permutation of itself); however, a string in L_1 must end in a 0, so permuting it will destroy this property unless the string is just a plain 0. Thus, $\textsc{Select}(L_1) = \{0\}$, which is clearly regular.

6.3. Regular languages are closed under the operation PERMUTE.
Solution: FALSE. For a counterexample, consider $L = (01)^*$. Any string in L contains an equal number of 0's and 1's, so, when we take all permutations of all strings in L, we get precisely the language $\{x \in \{0,1\}^* : x \text{ contains as many } 0\text{'s as } 1\text{'s}\}$. We have proved in class, using the pumping lemma, that this language is not regular. Thus $\textsc{Permute}(L)$ need not be regular even if L is.

6.4. Regular languages are closed under the operation SELECT.

Solution: FALSE. For a counterexample, consider $L' = \overline{(01)^*}$, with the complement being taken with respect to alphabet $\{0,1\}$. Clearly L' is regular. If a string x has an unequal number of 0's and 1's, then no matter how we permute it we will never land in the set $(01)^*$, i.e., every permutation of x is in L', i.e., $x \in \text{SELECT}(L')$. On the other hand, if a string x has as many 0's as 1's, then it *can* be permuted into the form $(01)^*$, so $x \notin \text{SELECT}(L')$. In short, we have shown that $\text{SELECT}(L') = \{x \in \{0,1\}^* : x$ contains an unequal number of 0's and 1's$\}$. Thus, $\text{SELECT}(L')$ is the complement of a non-regular language, whence it is itself non-regular.

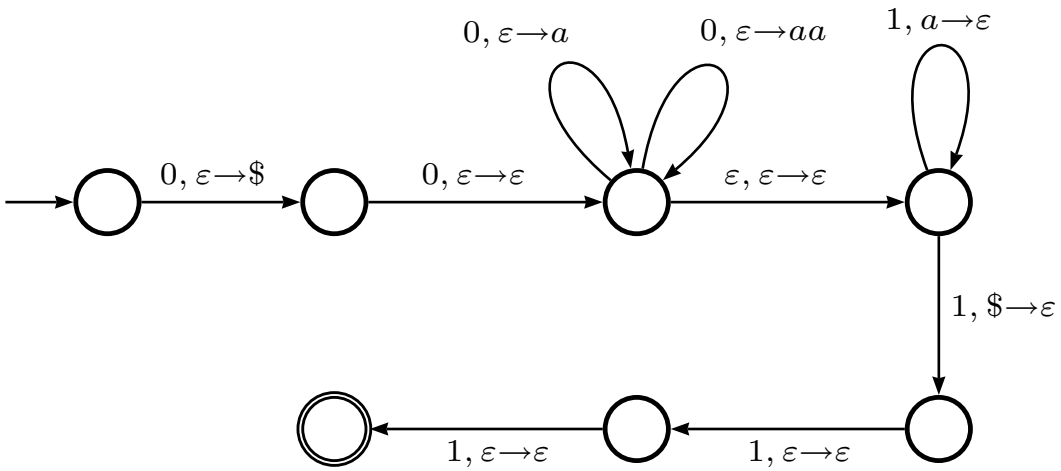
7. Draw a PDA for the language $\{0^i 1^j : i < j < 2i\}$. For clarity, keep your stack alphabet disjoint from $\{0,1\}$. Provide a brief justification (no need for a formal proof) that your PDA works correctly.

Solution: Let L be the given language. First, let us try to build a PDA for a slightly easier language: $L' = \{0^i 1^j : i \leq j \leq 2j\}$. The idea is to push either one or two a's (choose nondeterministically) onto our stack as we read the 0's in the input string, and then pop one symbol at a time as we read the 1's. If we empty the stack at the same time as when we finish reading the input, we may accept.

But for the language L, we must reject strings of the form $0^i 1^i$ and $0^i 1^{2i}$. To do so, observe that the shortest string in L is 00111 and that any string in L is therefore of the form

$$000^{i'} 1^{j'} 111$$

for some $i' \geq 0, j' \geq 0$, and $(i'+2) < (j'+3) < 2(i'+2)$. Since i' and j' are integers, the latter condition is equivalent to $i' \leq j' \leq 2i'$. Thus, strings in L are simply strings in L' with two 0's prepended and three 1's appended. With this in mind, we build our PDA:



8. Design a context-free grammar for the *complement* of the language $\{a^n b^n : n \geq 0\}$ over the alphabet $\{a,b\}$. Give brief explanations for the "meanings" of your variables (i.e. explain what strings are generated by each of your variable).

Solution: A string in the complement of $\{a^n b^n : n \geq 0\}$ is either of the form $\{a^i b^j : i > j \geq 0\}$ or of the form $\{a^i b^j : j > i \geq 0\}$ or not of the form $a^* b^*$, which means that it contains the substring ba. In the following grammar, the variable U generates strings of the third type,

while T generates strings of the form $a^n b^n$ to which we either prepend a string of one or more a's (generated by A) or append a string of one or more b's (generated by B).

$$\begin{aligned} S &\longrightarrow AT \mid TB \mid U \\ A &\longrightarrow Aa \mid a \\ B &\longrightarrow Bb \mid b \\ T &\longrightarrow aTb \mid \varepsilon \\ U &\longrightarrow VbaV \\ V &\longrightarrow Va \mid Vb \mid \varepsilon \end{aligned}$$