CSE596 **Problem Set 2 Answer Key** **Fall 2022**

(1) Prove that two of the following three languages are non-regular, via a Myhill-Nerode argument. For the regular one, give a regular expression. Here $\#a(x)$ denotes the number of occurrences of the character a in the string x, and more generally, $\#w(x)$ denotes the number of occurrences of the substring w in x. For example, #0010(00100100) = 2 even though the two occurrences of the substring 0010 overlap each other. Also, for two strings x, y of the same length, $x \oplus y$ denotes the bitwise exclusive-OR, e.g. $1011 \oplus 0010 = 1001$. All three languages are over the alphabet $\Sigma = \{0, 1\}$.

(i) $L_1 = \{x : \#0(x) \leq \#1(x)\}$.

(ii) $L_2 = \{x : \#01(x) = \#10(x)\}$.

(iii) $L_3 = \{xy : |x| = |y| \wedge x \oplus y = 1^{|x|}\}$.

($3 \times 12 = 36$ pts.)

Answers: L_2 is regular; the others are not. For L_2, the point is that for any binary string x, $\#01(x) = \#10(x)$ exactly when x begins and ends with the same character (or is the empty string). If the first char is 0, then every changeover to 1s involves one 01 substring, and changing back to 0s creates exactly one balancing 10 substring. For beginning and ending with 1 it is similar. Paying due attention to edge cases, a regular expression is

$$r_2 = 0(0 + 1)^*0 + 1(0 + 1)^*1 + 0 + 1 + \epsilon.$$

For both L_1 and L_3, we can choose $S = 0^*$, which is clearly infinite. Let any $x, y \in S$, $x \neq y$, be given. Then we can write $x = 0^m$, $y = 0^n$, where without loss of generality we may assert $m < n$. Take $z = 1^m$. Then $xz \in L_1$ and $xz \in L_3$, the latter because $0^m \oplus 1^m = 1^m$. But $yz \notin L_1$ because $\#0(yz) = n > m = \#1(yz)$, and $yz \notin L_3$ for these reasons: First, if $n + m$ is odd then there is no way to break yz into equal pieces. For $n + m$ being even, let us rewrite the definition of L_3 without a "symbol clash" as $L_3 = \{uv : |u| = |v| \wedge u \oplus v = 1^{|u|}\}$. Then the only way to break yz into equal-length pieces u, v makes u all-0s and includes at least one 0 into v. Those two 0s have an XOR value of 0, not 1, so $u \oplus v \neq 1^{|u|}$. Thus S is an infinite PD set for L_1 and also for L_3, so both L_1 and L_3 are non-regular.

(2) Now consider $L_4 = \{x : \#010(x) = 0 \wedge \#101(x) = 0\}$. Use the Myhill-Nerode technique to show that any DFA M such that $L(M) = L_4$ requires at least 6 states. Then design such a DFA M—ideally showing how your proof guided you to it (or vice-versa). Finally explain why you can basically "collapse" M into a generalized NFA with only 2 states s, f such that

$$L(M) = L_{s,s} \cup L_{s,f} \cup L_{f,s} \cup L_{f,f},$$

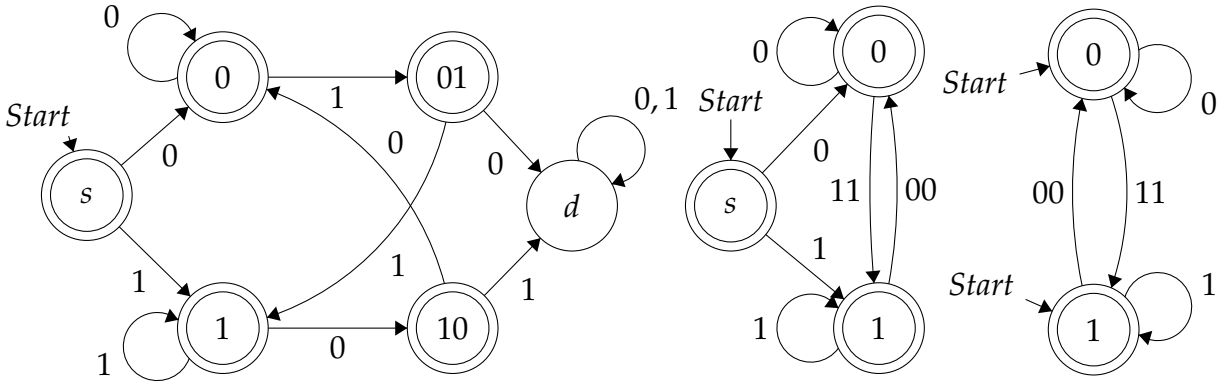
and use that to give a regular expression for L_4. (12 + 6 + 9 = 27 pts.)

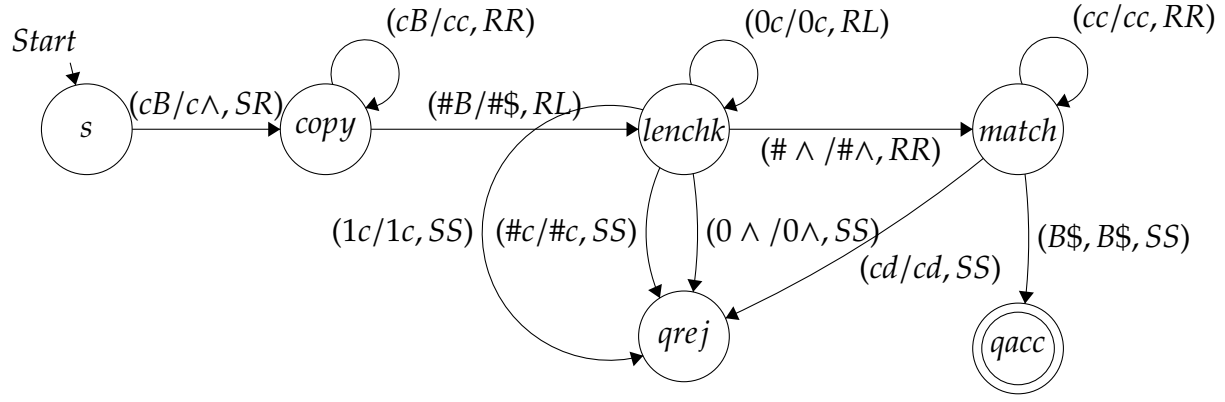
Answer: The strings on the way to building 010 and/or 101 are $\{\epsilon, 0, 1, 01, 10\}$, plus either 010 or 101 causes a "dead condition." Let's choose 010, so $S = \{\epsilon, 0, 1, 01, 10, 010\}$. Now 010 is separated from the other five strings by ϵ, since $010 \notin L_4$ whereas the other five

strings all belong to L_4. So we need only separate those five strings from each other. It was fine to do so by itemizing all $\binom{5}{2} = 10$ pairs, but here is a shortcut: For those pairs x, y with $|x| < |y|$, take z such that $yz = 010$ or $yz = 101$; then $yz \notin L_4$ while $|xz| \leq 2$ so xz belongs to L_4. This leaves just the pair $x = 0$, $y = 1$, which is separated by $z = 10$ (or by $z = 01$) and the pair $x = 01$, $y = 10$, which is separated by $z = 0$ (or by $z = 1$). Thus S is a PD set of size 6.

This means every DFA M such that $L(M) = L_4$ must process the strings in S to different states, including 010 (and 101) going to a dead state—with the other states accepting. This pretty much dictates the design shown at left in the figure. Now to find a regular expression, we can first delete the dead state since it can never be used in an accepting computation. Then the states q_{01} and q_{10} have only one incoming and one outgoing arc, so they can be elided to produce the second machine. Then because the start state is accepting, nothing is disturbed by deleting it but allowing either q_0 or q_1 to be the start state. This yields the hinted two-state GNFA N with:

$$\begin{aligned} L(M) = L(N) &= L_{0,0} \cup L_{0,1} \cup L_{1,0} \cup L_{1,1} \\ &= (0 + 111^*00) * (\epsilon + 111^*) + (1 + 000^*11)^*(\epsilon + 000^*). \end{aligned}$$





(3) Design a two-tape deterministic Turing machine M_2 that recognizes the language

$$L_3 = \{x\#0^k\#y : x, y \in \{0,1\}^*, x = y \wedge |x| = k\}.$$

Here $\Sigma = \{0, 1, \#\}$ but the # character is only allowed as a marker to divide the input string w into thirds. Your M_2 should run in $O(n)$ time where $n = |w|$; note that any accepted string gives $n = 3k + 2$ with k as above. (A well-commented arc-node drawing is fine; if you use

the Turing Kit, please take a screenshot since its own Postscript-based print feature is old and may be wonky.)

Answer shown above. It needs some comments, first that 'c' and 'd' are "wildcards" standing for 0 or 1. In the "match" state, the interpretation is that the more-specific arc ($cc/cc, RR$) where the chars are equal takes precedence over the most-general case ($cd/cd, SS$), leaving "cd" in the latter to apply to those cases where the characters are different. Some arcs going to the reject state are not shown—they are cases in which the input has fewer than two or more than two '#' characters. When there are exactly two '#' characters, the input has the form $x\#u\#y$. The "copy" state copies x to the second tape as $\wedge x\$$ where the endmarkers $\wedge$ and \$ make the next stages easier to code and interpret. While moving the second tape head leftward—and *not* writing blanks which is where M_2 is not a PDA—it checks that u has no 1 and that $|u| = |x|$. The condition $|u| < |x|$ is caught by the tape-1 head seeing the second # while the tape-2 is still reading 0s and 1s that were copied; the condition $|u| > |x|$ is caught by the tape-2 head reaching the left-endmarker $\wedge$ (which was placed down in the initial step) while the input head is still reading 0s. When $u = 0^{|x|}$ is verified, the heads are in perfect position to check $y = x$, which is then the condition for M_2 to accept. Not only does M_2 run in linear time, it moves its input head right in each step except the single initial and final steps, which means it runs *in real time*.

Then argue as best you can that every single-tape TM M_1 such that $L(M_1) = L_3$ requires $\Omega(n^2)$ time. Since n is linear in k, it may help to think of this as $\Omega(k^2)$ time. Start by showing that $S = \{0,1\}^k$ is PD fopr L_3. Then argue that this means k bots of information must somehow cross the middle 0^k part in order to decide $y = x$ correctly. Finally reckon how much total time M_1 must spend in that middle region, noting that M has a fixed number $r = |Q|$ of states but k can grow. (You may consult last year's key https://cse.buffalo.edu/ regan/cse596/F18/CSE596ps3key.pdf for a related problem where the one-tape time is $\Omega(n \log n)$, but your answer does not have to use the Australian imagery. 18 + 18 = 36 pts.)

Answer: For any k, let any distinct $x_1, x_2 \in \{0,1\}^k$ be given. Take $z = \#0^k\#x_1$. Then $x_1 z \in L_3$ since its "y part" equals x_1 and its middle has the right length k, but $x_2 z \notin L_3$. In particular, it is not in the "slice" of L_3 that deals with this value of k. So S is PD for L_3. The intuititve meaning of this is that the "x part" requires 2^k distinctions to be made at the other end of the tape, which is captured by saying that (at least) k bits of information need to be transported from left to right.

It was acceptable to say that anytime the single-tape TM $M_1 = (Q, \Sigma, \Gamma, \ldots)$ crosses the second # sign left-to-right, it carries in only the information in its current state. So it can carry at most $q = \log_2(|Q|)$ bits at a time. Thus the computation needs k/q trips from the "x" part to the "y" part. Since there are k cells in-between, this takes order-of $k \cdot k/q = \Omega(k^2)$ steps overall. This is hence a lower bound on the running time.

If this last part seems hand-wavy to you, then you may appreciate how *Kolmogorov complexity* rigorizes the argument. This will be done as a separate post after the first exam. Briefly put, if M could work in time $t = o(k^2)$ then the average time it spends among the k cells in the middle part is $t/k = o(k)$. Some cell j is at most average. Every string $y \in \{0,1\}^k$ could then be specified by giving just the sequence of states M is in on its at most t/k crossings—since the "$x\#0^k\#$" part would have to automatically match it for M to accept. Then every string $y \in \{0,1\}^k$ would be specifiable in $< k$ bits, which is impossible.