Midterm 1: October 1, 2018

Real name: 

NetID: 

Gradescope name: 

Gradescope email: 

- This is a closed-book, closed-notes, closed-electronics exam. If you brought anything except your writing implements, put it away for the duration of the exam. In particular, you may not use any electronic devices other than those that are medically necessary.

- Please clearly print your real name, your university NetID, your Gradescope name, and your Gradescope email address in the boxes above. If you are using your real name and your university email address on Gradescope, you do not need to write everything twice. We will not scan this page into Gradescope.

- Please also print only the name you are using on Gradescope at the top of every page of the answer booklet, except this cover page. These are the pages we will scan into Gradescope.

- Please do not write outside the black boxes on each page; these indicate the area of the page that the scanner can actually see.

- This answer booklet is double-sided!

- If you run out of space for an answer, feel free to use the scratch pages at the back of the answer booklet, but please clearly indicate where we should look.

- Please read the entire exam before writing anything. There are seven numbered problems and each problem is worth 10 points.

- You have 150 minutes.

- Proofs are required only if we specifically ask for them.
Describe a DFA for the language defined below.

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ contains 01 as a substring and } |w| \text{ is even} \} \]

Your DFA must have at most 6 states. Briefly explain the states of the DFA. You may either draw the DFA or describe it formally in tuple notation. If you specify it via tuple notation, the states \( Q \), the start state \( s \), the accepting states \( A \), and the transition function \( \delta \) must be clearly specified.
Assume $\Sigma = \{0, 1\}$. Recall that a block of 1’s in a string is a maximal non-empty substring of 1’s; the blocks of 1’s are underlined in 0100110111101. Describe a regular expression for the language defined below.

$L = \{w \in \{0, 1\}^* \mid w$ has at most one block of 1’s of even length$\}$.

The strings 01110101 and 01110110 are in the language but 1101111 and 11011001001111 are not. Even length blocks of 1s are underlined. Briefly explain your regular expression.
Given a language $L$ over alphabet $\Sigma$ recall that \textsc{Prefix}(L) is the language defined over $\Sigma$ as the collection of all prefixes of strings in $L$. Formally, $\text{PREFIX}(L) = \{ u \mid \exists v \in \Sigma^*, uv \in L \}$. In this problem, assuming that $L$ is regular, you will derive an algorithm that generates a regular expression $r'$ for \textsc{Prefix}(L) from a regular expression $r$ for $L$. No justification necessary.

- For each of the base cases write a regular expression $r'$ for \textsc{Prefix}(L($r$)).

  (i) $r = \emptyset$: $r' = \emptyset$

  (ii) $r = \varepsilon$: $r' = \varepsilon$

  (iii) $r = a, a \in \Sigma$: $r' = a + \varepsilon$

- Assume $r = r_1 + r_2$ and that $r'_1$ and $r'_2$ are regular expressions for \textsc{Prefix}(L($r_1$)) and \textsc{Prefix}(L($r_2$)) respectively. Write a regular expression $r'$ for \textsc{Prefix}(L($r$)) in terms of $r_1, r_2, r'_1, r'_2$.

  $$r' = r'_1 + r'_2$$

- Assume $r = r_1 r_2$ and that $r'_1$ and $r'_2$ are regular expressions for \textsc{Prefix}(L($r_1$)) and \textsc{Prefix}(L($r_2$)) respectively. Write a regular expression $r'$ for \textsc{Prefix}(L($r$)) in terms of $r_1, r_2, r'_1, r'_2$.

  $$r' = \left( r'_1 \right) + \left( r'_2 \right)$$

- Assume $r = r_1^*$ and that $r'_1$ is a regular expression for \textsc{Prefix}(L($r_1$)). Write a regular expression $r'$ for \textsc{Prefix}(L($r$)) in terms of $r_1, r'_1$.

  $$r' = \left( r'_1 \right) + \left( r_1 \right)$$
Prove that the language \( \{a^i b^j c^k \mid i + j < k \} \) over the alphabet \( \{a, b, c\} \) is not regular.
Describe a CFG for the language \(\{a^ib^j c^k \mid i + j < k\}\). In order to get full credit you should briefly explain how your grammar works, and the role of each non-terminal.

\[
S \rightarrow \text{more } c \text{'s than } a \text{'s, } b \text{'s} \\

b^m c^n \rightarrow x \rightarrow b x c \mid c \\
\begin{align*}
a^m b^m c^n c^m & \rightarrow y \rightarrow a y c \mid x \\
c^n & \text{ n>1} \\
z & \rightarrow c \mid c z \\
S & \rightarrow y z \\
a^m b^m c^{m+n+k} & \text{ k>0}
\]
Let $G_1, G_2, G_3$ be context free grammars for languages $L_1, L_2, L_3$ respectively. Let $G_1 = (V_1, T, P_1, S_1)$ and $G_2 = (V_2, T, P_2, S_2)$ and $G_3 = (V_3, T, P_3, S_3)$ and assume that the non-terminal symbols $V_1, V_2, V_3$ are mutually disjoint (that is, they don’t share any symbols). Describe a CFG $G = (V, T, P, S)$ for the language

$$L = L_1 + L_2L_3^*.$$ 

*Hint:* You may want to recall how we proved the closure properties of CFGs under union, concatenation and Kleene star.
Bitstrings are another name for strings over the binary alphabet \{0, 1\}. Given a bitstring \(w\) let \(\text{flip}(w)\) be the string obtained by “flipping” each bit of the string, that is changing a 0 to 1 and a 1 to a 0. For example \(\text{flip}(010110) = 101001\). Given a language \(L \subseteq \{0, 1\}^*\) we define \(\text{flip}(L) = \{\text{flip}(w) \mid w \in L\}\). As an example, if \(L = \{0, 0110\}\) then \(\text{flip}(L) = \{1, 1001\}\). Given a language \(L \in \{0, 1\}^*\) we define \(\text{flipsuffix}(L)\) as follows.

\[
\text{flipsuffix}(L) = \{u \text{flip}(v) \mid uv \in L\}.
\]

As an example, if \(L = \{0, 0110\}\) then \(\text{flipsuffix}(L) = \{0, 1, 0110, 0111, 0101, 0001, 1001\}\) where the underlined segments indicate the flipped suffixes.

(a) Given a DFA \(M = (Q, \{0, 1\}, \delta, s, A)\) for a regular language \(L\), describe a DFA or NFA that accepts the language \(\text{flip}(L)\).

(b) Given a DFA \(M = (Q, \{0, 1\}, \delta, s, A)\) for a regular language \(L\), describe a DFA or NFA that accepts the language \(\text{flipsuffix}(L)\). Note that \(\text{flipsuffix}(L)\) is not necessarily same as \(\text{PREFIX}(L) \cdot \text{flip}(\text{SUFFIX}(L))\). The previous part is to help you think about this second part. If you are confident about the solution to this part you can skip the previous part and get full credit.
This page for extra work.
This page for extra work.
This page for extra work.
This page for extra work.
This page for extra work.