CS 452/508: Formal Languages and Automata
Assignment 5
Due Date: February 28, 2019

Instructor: Robert Robere
Spring 2019

As usual, the questions below marked “Extra Practice” are for your own interest, and will not be marked. Note. Question 1a is now a bonus question, after discussions with several students.

1. Recall that on Assignment 2 we showed the language $L$ defined by:

   $$L = \{ x \in \{a, b\}^* \mid x \text{ contains an equal number of occurrences of } ab \text{ and } ba \}$$

   is regular. Consider the language

   $$L' = \{ x \in \{a, b, c\}^* \mid x \text{ contains an equal number of occurrences of } ab \text{ and } ba \}.$$ 

   (a) (NOW BONUS QUESTION.) Prove or disprove that there is a $p$ such that for any string $x \in L'$ with length at least $p$, there is a partition $x = uvw$ into strings $u, v, w$ such that

   i. $uvw \in L$ for all $i \geq 0$.
   ii. $|uv| \leq p$
   iii. $|v| > 0$.

   That is, $L'$ satisfies the pumping property.

   (b) Show that $L'$ is not regular. (Hint. Use the closure properties of regular languages!)

   Solution. Let $A = (abc)^*c(cba)^*$, and note that $A$ is clearly regular. Assume by way of contradiction that $L'$ is regular, then, since regular languages are closed under intersection we have that

   $$A \cap L' = \{(abc)^nc(cba)^n : n \geq 0\}$$

   is also regular. However, we can prove $A \cap L'$ is not regular by the pumping lemma, which give us the desired contradiction.

   Let $p$ be the pumping length guaranteed by the pumping lemma for $A \cap L'$, and let $x = (abc)^pc(cba)^p$. By the pumping lemma there are three strings $u, v, w$ such that $x = uvw$ such that the properties of the pumping lemma hold. Since $|uv| \leq p$ it follows that $v$ is
a substring of \((abc)^p\), and by the pumping lemma we have \(uv^2w \in A \cap L'\). However, since \(v\) is non-empty and a substring of \((abc)^p\), it clearly follows that \(uv^2w \notin A \cap L'\), as duplicating any substring of \((abc)^p\) will either: (1) cause a mismatch with \((cba)^p\) (for example, if \(v = (abc)^i\) for some \(i > 0\)), or (2), the beginning of the string will not be of the form \((abc)^k\) for some \(k \geq 0\). Thus \(A \cap L'\) is not regular, and since \(A\) is regular, \(L'\) cannot be regular.

2. Let us say a context-free grammar \(G\) is simple if all rules are of the following form:

- \(A \rightarrow aB\) for non-terminals \(A, B\) and terminal symbols \(a\), or
- \(A \rightarrow a\) for non-terminal \(A\) and terminal \(a\), or
- \(A \rightarrow \varepsilon\).

Prove that a language \(L \subseteq \Sigma^*\) is generated by a simple grammar if and only if \(L\) is a regular language.

**Solution.** There is a one-to-one correspondence between simple grammars and NFAs. Given a simple grammar \(G\) over alphabet \(\Sigma\) and with set of non-terminals \(V\), consider the following NFA \(M\). The states of \(M\) are \(Q = V \cup \{q^*\}\), where \(q^*\) is a special accept state, and the start state of \(M\) is \(S\), the start variable of the grammar. The (only) accept state of \(M\) is \(q^*\). Finally, for each rule of the grammar of the form \(A \rightarrow aB\) we add a transition from \(A\) to \(B\) labelled with \(a\), and for each rule of the grammar of the form \(A \rightarrow a\) where \(a \in \Sigma \cup \{\varepsilon\}\) we add a transition from \(A\) to \(q^*\) labelled with \(a\). Clearly if a string \(x\) is in the language of \(G\) then \(x\) is also accepted by the NFA by following the sequence of transitions in \(M\) that correspond to the sequence of rule applications in \(G\) used to derive \(x\), and the converse is also clear.

On the other hand, given an NFA \(M = (\Sigma, Q, q_0, \delta, F)\) we can construct a simple grammar \(G\) by “reversing” the above transformation. First transform \(M\) into a new NFA \(M'\) with a new unique accept state \(q^*\) by adding \(\varepsilon\)-transitions from each \(q \in F\) to \(q^*\). The variables of the grammar \(G\) will be \(Q\), the states of \(M\) (note that \(q^*\) is not included), and the starting variable is \(q_0\). For each pair of states \(q_i, q_j\) with \(q_j \neq q^*\) and for each transition from \(q_i\) to \(q_j\) labelled with terminal symbol \(a\) we add the rule \(q_i \rightarrow aq_j\) to the rules of the grammar. Finally, for each \(\varepsilon\)-transition from any state \(q\) to the final state \(q^*\) we add the rule \(q \rightarrow \varepsilon\). The correctness of this transformation is immediate by the same argument made before.

3. Give a context-free grammar for the language

\[
\{a^ib^jc^k \mid i = j \text{ or } j = k \text{ where } i, j, k \geq 0\}.
\]

Is your grammar ambiguous?

**Solution.** We give a grammar for \(\{a^n b^n c^n : n, i \geq 0\}\) and a grammar for \(\{a^i b^n c^n : n, i \geq 0\}\) and then create a grammar for the union of the two. A grammar for the first language is

\[
S_1 \rightarrow A_1 U_1, A_1 \rightarrow aA_1 b|\varepsilon, U_1 \rightarrow cU_1|\varepsilon,
\]

where \(S_1\) is the start variable. A grammar for the second language is

\[
S_2 \rightarrow U_2 A_2, A_2 \rightarrow bA_2 c|\varepsilon, U_2 \rightarrow aU_2|\varepsilon,
\]

\end{verbatim}
where $S_2$ is the start variable. The grammar for the union of the two languages is therefore given by

\[
S \rightarrow S_1|S_2 \\
S_1 \rightarrow A_1U_1, A_1 \rightarrow aA_1b|\varepsilon, U_1 \rightarrow cU_1|\varepsilon \\
S_2 \rightarrow U_2A_2, A_2 \rightarrow bA_2c|\varepsilon, U_2 \rightarrow aU_2|\varepsilon.
\]

This grammar is clearly ambiguous. The string $abc$ can be generated in the following two ways:

\[
S \Rightarrow A_1U_1 \Rightarrow aA_1bU_1 \Rightarrow abU_1 \Rightarrow abc \\
S \Rightarrow U_2A_2 \Rightarrow aA_2 \Rightarrow abA_2c \Rightarrow abc.
\]

4. Using the algorithm discussed in class, minimize the following DFA, which computes the language

\[
L = \{x \in \{0, 1\}^* \mid x \text{ has an even number of 0s and a 1 after each 0}\}.
\]

A complete answer will contain:

(a) A state diagram of the minimized DFA.

(b) A list of the equivalence classes of the relations $\equiv_0$, $\equiv_1$, $\equiv_2$, \cdots obtained by the algorithm until a fixed point is reached.

![State Diagram](image-url)
Solution. Here is a state diagram of the minimized DFA.

And here are the equivalence relations produced by the algorithm until a fixed point is reached.

\[\equiv_0: \{q_0\}, \{q_1, q_2, q_3, q_4, q_5\}\]
\[\equiv_1: \{q_0\}, \{q_1\}, \{q_2, q_3, q_4, q_5\}\]
\[\equiv_2: \{q_0\}, \{q_1\}, \{q_3\}, \{q_2, q_4, q_5\}\]
\[\equiv_3: \{q_0\}, \{q_1\}, \{q_3\}, \{q_4\}, \{q_2, q_5\}\]
\[\equiv_4: \{q_0\}, \{q_1\}, \{q_3\}, \{q_4\}, \{q_2, q_5\}\]