Homework 4 Solutions

COMS W3261, Summer B 2021

This homework is due **Monday, 7/26/2021, at 11:59PM EST**. Submit to GradeScope (course code: X3JEX4).

Grading policy reminder: LaTeX is preferred, but neatly typed or handwritten solutions are acceptable. Your TAs may dock points for indecipherable writing. Proofs should be complete; that is, include enough information that a reader can clearly tell that the argument is rigorous.

The tool http://madebyevan.com/fsm/ may be useful for drawing finite state machines.

If a question is ambiguous, please state your assumptions. This way, we can give you credit for correct work. (Even better, post on Ed so that we can resolve the ambiguity.)

1 Problem 1 (10 points)

1. (3 points). Show that the context-free grammar G_1, given by the rules

$$\begin{aligned} S &\to 0A10 \mid B10 \\ B &\to A0 \mid B1 \\ A &\to 00 \mid \varepsilon \end{aligned}$$

is ambiguous by finding a string with two different leftmost derivations. ($V = \{S, A, B\}$ and $\Sigma = \{0, 1\}$.)

Consider the string 010. It can be derived as follows:

$$S \Rightarrow 0A10 \Rightarrow 010,$$

using the first rule for replacing S and the second rule for replacing A, or

$$S \Rightarrow B10 \Rightarrow A010 \Rightarrow 010,$$

using the second rule for replacing S, the first rule for replacing B and the second rule for replacing A.

2. (3 points). Identify the language generated by G_1 above. (It's fine to write this language as a simple regular expression.)

First, we observe that the variable A produces only strings of terminals. Substituting A for 00 and for ε wherever it occurs on the righthand side, we get the following equivalent grammar G_2:

$$\begin{aligned} S &\to 00010 \mid 010 \mid B10 \\ B &\to 000 \mid 0 \mid B1 \end{aligned}$$

Moreover, we observe that the rules that substitute S for the strings 00010 and 010 are redundant, as these can be created by first using the substitution rule $S \to B10$. This yields the equivalent grammar G_3:

$$\begin{aligned} S &\to B10 \\ B &\to 000 \mid 0 \mid B1 \end{aligned}$$

The first rule ensures all strings that can be derived from S end in the substring 10, and the first two rules for replacing B ensure that the string always begins with the substring 000 or the substring 0. The language is thus $(000 \cup 0)1^*10$.

3. (4 points). Design a context-free grammar that generates the same language as G_1 but which is not ambiguous (i.e., no string admits two distinct leftmost derivations.)

The grammar G_3 above has this property. To see this, observe that any string in the language $(000 \cup 0)1^k10$ for any $k \geq 0$ can be produced only by a derivation that begins with $S \Rightarrow B10$, uses the rule $B \to B1$ exactly k times, and then ends with either $B \Rightarrow 000$ or $B \Rightarrow 0$ depending on the number of leading 0s.

2 Problem 2 (8 points)

1. (3 points). Identify the language generated by the context-free grammar G_2 below. (It's fine to describe the language in words or symbols or write a regular expression.)

$$\begin{aligned} S &\to S\#S \mid A \\ A &\to 10 \mid 0A \end{aligned}$$

The first rule $S \to S\#S$ ensures that any string in the language consists of substrings in $\{0,1\}^*$ separated by the # symbol. Each of these substrings is generated by using the rule $A \to 0A$ some number of times before using the rule $A \to 10$ to complete a sequence of terminals. Thus strings derived from A match the regular expression 0^*10. The language is captured by the regular expression

$$(0^*10\#)^*0^*10.$$

2. (5 points). Use the procedure outlined in class to convert G_2 to an equivalent pushdown automata P. (To simplify the state set, you may write a transition function that pushes strings. For example, you might define $\delta(q_1, a, b) = \{(q_2, s)\}$ for some string $s \in \Gamma^*$ as long as you mention that the process of pushing strings implicitly requires some additional states. You may write your answer as a 6-tuple or as an equivalent state diagram. Either way, explain the conversion process.)

We will define a pushdown automaton $P = (Q, \Sigma, \Gamma, \delta, q_{start}, F)$ that recognizes the language G_2. We'll build P according to the procedure outlined in class, so P will have three 'main' states: q_{start}, from which we push the string $S\$$, q_{loop}, from which we implement substitution rules, and q_{accept}, where our computation proceeds after we've finished substituting and can pop the final \$ off the stack. The components of P are defined as follows:

- $Q = \{q_{start}, q_{loop}, q_{accept}\} \cup E$, where E denotes the set of additional states that are implicitly necessary to push strings onto the stack.
- $\Sigma = \{0, 1, \#\}$.
- $\Gamma = \{S, A, 0, 1, \#, \$\}$.
- $F = \{q_{accept}\}$.

q_{start} is the start state, so it remains to specify the transition function δ. All unspecified transitions map to the empty set $\emptyset$.

First, we add transitions

$$\begin{aligned} \delta(q_{start}, \varepsilon, \varepsilon) &= \{(q_{loop}, S\$)\} \\ \delta(q_{loop}, \varepsilon, \$) &= \{(q_{accept}, \varepsilon)\}. \end{aligned}$$

These ensure that we begin substituting with $S\$$ on our stack and enter an accept state when we pop \$ (subsequently accepting if we have finished the input string.) It remains to add

$\varepsilon, A \to w$ transitions for every rule $A \to w$ in G_2 (to substitute variables) and $a, a \to \varepsilon$ transitions for every terminal $a \in \Sigma$ (to match the input string with the produced string of terminals). For G_2 this results in the following ruleset:

$$\begin{aligned}
\delta(q_{loop}, \varepsilon, S) &= \{(q_{loop}, S\#S), (q_{loop}, A)\} \\
\delta(q_{loop}, \varepsilon, A) &= \{(q_{loop}, 10), (q_{loop}, 0A)\} \\
\delta(q_{loop}, 1, 1) &= \{(q_{loop}, \varepsilon)\} \\
\delta(q_{loop}, 0, 0) &= \{(q_{loop}, \varepsilon)\} \\
\delta(q_{loop}, \#, \#) &= \{(q_{loop}, \varepsilon)\}.
\end{aligned}$$

3 Problem 3 (8 points)

1. (3 points). Identify the language generated by the context-free grammar G_3 below.

$$\begin{aligned} &S \to A \mid B \\ &A \to 0A00 \mid \# \\ &B \to 0B000 \mid \# \end{aligned}$$

If a derivation from S begins with the production rule $S \to A$, it then uses the rule $A \to 0A00$ exactly k times for some $k \geq 0$ and then ends with the rule $A \to \#$, thus producing a string in the language

$$\{0^k\#0^{2k} \mid k \geq 0\}.$$

Similarly, any derivation that begins with the production rule $S \to B$ derives a string in the language

$$\{0^k\#0^{3k} \mid k \geq 0\}.$$

G_3 thus generates the language

$$L = \{0^k\#0^{2k} \mid k \geq 0\} \cup \{0^k\#0^{3k} \mid k \geq 0\}.$$

2. (5 points). Prove that the language of G_3 is nonregular using the pumping lemma.

Assume for contradiction that the language L is regular. By assumption, L satisfies the pumping lemma and there must exist a pumping length p such that for all $s \in L$, $|s| \geq p$, s can be divided into substrings x, y, and z such that $|y| > 0$, $|xy| \leq p$, and $xy^iz \in L$ for all $i \geq 0$.

Consider the string $s = 0^p\#0^{3p}$, which has length $4p + 1$ and is in the language. We show that this string cannot be pumped.

Consider any partition of s into substrings x, y, and z satisfying $|xy| \leq p$ and $|y| > 0$. By the assumption that $|xy| \leq p$, y is part of the first substring of $0's$. Thus we have

$$xy^0z = xz = 0^{p-|y|}\#0^{3p}.$$

Because $|y| > 0$, we have that $3p > 3(p - |y|) > 2(p - |y|)$, and thus it is not true that $0^{p-|y|}\#0^{3p} = 0^k\#0^{3k}$ or $0^{p-|y|}\#0^{3p} = 0^k\#0^{2k}$ for any $k \geq 0$. Thus xz is not in the language and the pumping lemma fails. Because the assumption that L is regular leads to a contradiction, L is nonregular.

4 Problem 4 (14 points)

1. (3 points). Prove that the class of context-free languages is closed under union. (Hint: We want to show that the union of any two context-free languages is context-free; i.e., for any two languages described by context-free grammars, their union can also be described by a context-free grammar.)

   Let L_1 and L_2 be two context-free languages. Let $G_1 = (V_1, \Sigma_1, R_1, S_1)$ and $G_2 = (V_2, \Sigma_2, R_2, S_2)$ be context-free grammars that describe L_1 and L_2, respectively. We show a context-free grammar that describes $L_1 \cup L_2$.

   Consider $G_3 = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R_3, S)$, where $R_3 = R_1 \cup R_2 \cup \{S \to S_1, S \to S_2\}$. The new ruleset ensures that S derives either S_1 or S_2, from which two variables can be derived precisely the strings in $L_1 \cup L_2$.

2. (3 points). Prove that the class of context-free languages is closed under concatenation.

   Let L_1 and L_2 be two context-free languages. Let $G_1 = (V_1, \Sigma_1, R_1, S_1)$ and $G_2 = (V_2, \Sigma_2, R_2, S_2)$ be context-free grammars that describe L_1 and L_2, respectively. We show a context-free grammar that describes $L_1 L_2$.

   Consider $G_3 = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R_3, S)$, where $R_3 = R_1 \cup R_2 \cup \{S \to S_1 S_2\}$. The new ruleset ensures that S derives $S_1 S_2$, from which string we can derive any string generated by G_1 concatenated with any string generated by G_2.

3. (3 points). Prove that the class of context-free languages is closed under star.

   Let $G_1 = (V, \Sigma, R, S)$ be a context-free grammar describing the language L_1. We show a context-free grammar that describes L_1^*.

   Consider $G = (V, \Sigma, R', S)$, where $R' = R_1 \cup \{S \to \varepsilon, S \to SS\}$. The new ruleset ensures that S derives any string that matches the pattern S^*, from which we can derive any string in L_1^*.

4. (5 points). Provide a new proof that every regular language is context-free by showing how to convert any regular expression into an equivalent context-free grammar. (Hint: we defined a regular expression inductively using three base cases and three recursive cases corresponding to regular operations. It suffices to explain how each of these cases can be converted into an equivalent CFG.)

   Our inductive definition of the regular expressions began with the statement that $\emptyset$, ε, and a for any $a \in \Sigma$ were valid regular expressions.

   - The language of the grammar $G_\emptyset = (\{S\}, \emptyset, \emptyset, S)$ (and any other grammar that does not generate any terminal string) is the empty language $\emptyset$.

     (Note after Ed discussion: What values are appropriate for the ruleset R depends on our precise formal definition of R. Sipser is slightly ambiguous on this point, so we will give credit for any reasonable answer, such as $R = \{S \to S\}$.)

   - The language of the grammar G_ε with the single rule $S \to \varepsilon$ generates the language containing the empty string $\{\varepsilon\}$.

- For any symbol a in a given alphabet Σ, the language of the grammar G_a with the single rule $S \to a$ is $\{a\}$.

The other three cases in our definition told us that for any regular expressions R_1 and R_2, $R_1 \cup R_2$, R_1R_2, and R_1^* were also regular expressions. Given CFGs G_1 and G_2 for R_1 and R_2, we can recognize the languages $R_1 \cup R_2$, R_1R_2, and R_1^* using the constructions in parts 1-3 of this problem.

Thus we can (recursively) convert a regular expression matching any of the six cases in our definition of the regular expressions to a CFG that recognizes the same language.