CSE322: Formal Models in Computer Science Spring 2006

Partial Solutions to Sample Final

This handout has (partial) solutions to some of the problems in the sample final exam that was handed out in class on Friday, May 26. I have not given solutions to any of the "mechanical" questions (like converting CFG to PDA). Also the solutions are terse– they are meant to give you the gist of the solution and should not be considered as "model" solutions. In particular, you can (and most probably will) lose credit in the final exam if your answers are not fleshed out more.

---

1. (True/False with justification questions)

   (a) **False**. The PDA could just push a "dummy" symbol onto the stack and never really use it in its' computation.

   (b) **True**. Use the pumping lemma for regular languages (as the language is infinite, it will have an s in the language such that $|s| \geq p$, where p is the pumping length).

   (c) **False**. The set of syntactically correct programs can be generated by a context-free grammar. Checking whether an input w can be generated by a CFG G is exactly the same as deciding A_{CFG}, which is decidable.

   (d) **False**. Let $R = \emptyset$. Then $L = \emptyset$ which is regular (and hence, a CFL). Now K can be any arbitrary language.

   (e) **True**. This is just the contra-positive of the closure of intersection of a CFL with a regular language.

   (f) **True**. The statement is "equivalent" to the halting problem. In particular, if the statement is false, then there would exist a decider (call it D) that can tell given an arbitrary TM P and w, whether P loops on w or not. Given such a decider, one can construct a decider for A_{TM} as follows. On input $\langle P, w \rangle$, first run the input on D. If D says yes, then simulate P on w and return the simulation's answer , otherwise reject. Of course, A_{TM} is undecidable and hence, we have a contradiction.

   (g) **True**. The number of possible stack arrangements is finite (in fact, bounded by $(|\Sigma| + 1)^{100}$). Thus, one can "encode" all the possibilities in the states of a (huge) DFA.

   (h) **True**. Just "reverse" the transitions.

   (i) **False**. A_{CFG} is decidable.

   (j) **True**. There are languages that are context-free but not regular.

2. (Classifying languages)

   (a) **B**. Here is a CFG for the language

   $$S \rightarrow aSbb \mid b$$

   To show it is not regular, use pumping lemma with $s = a^p b^{2p+1}$.

   (b) **A**. The language is the same as $\{a^m \mid m \bmod 3 \equiv 1\}$.

   (c) **A**. One can express this language as $L_0^a \circ L_0^b \cup L_1^a \circ L_1^b \cup L_2^a \circ L_2^b$, where $L_i^a = \{a^n \mid n \equiv i \bmod 3\}$, $L_i^b = \{b^n \mid n \equiv i \bmod 3\}$ $(i = 0, 1, 2)$.

   (d) **B**. The language is the intersection of a CFL language $(\{xcx^R \mid x \in \{a, b\}^*\})$ and a regular language $((b^*ab^*ab^*)^*c(b^*ab^*ab^*)^*$. To prove this language is not regular use the pumping lemma with $s = a^{2p}ca^{2p}$.

(e) **B**. The PDA for this language works as follows. Push in all the a's and b's that come in and match each of them with the c's. To prove that this language is not regular use the pumping lemma with $s = a^p b^p c^{2p}$.

(f) **B**. The PDA for this language non-deterministically decides which of $k \neq i$ or $k \neq j$ is true and then goes on to check that part. To prove that this language is not regular, use the pumping lemma with $s = a^p b^{p!+p} a^{p!+p}$ and pump $i = p!/|y| + 1$ times.

(g) **C**. An intuitive argument is that to check both conditions, one needs two stacks.

(h) **B**. This language is $L \circ L$, where $L = \{0^n 1^n \mid n \geq 0\}$. For non-regularity use pumping lemma with $s = 0^p 1^p$.

(i) **C**. Again one needs two stacks to do this. Consider the following natural one stack PDA candidate (that does not work). First push in all the a's and then the b's. However, to match the a's one has to pop off all the b's, which will not allow the matching of b's later.

(j) **A**. This language is the set of all non-empty strings. To see this note that any $z \in \{a, b\}^*$, with $z \neq \epsilon$ can be written as xy, where $x = \epsilon$ and $y = z$.

3. Let p_i denote the i^{th} prime number and note that the sequence $p_1, p_2, \ldots$ is infinite. Now for using the Myhill-Nerode theorem, choose $x_i = 0^{p_i}$ and pick $z_{ij} = 1^{p_i}$. To finish the argument use the fact that for any $i \neq j$, $\gcd(p_i, p_j) = 1$.

4. I will skip parts (a) and (b) as they are "mechanical". For part (c), G generates all strings over $\{a, b\}$ that have twice as many a's as b's.

5. Here is the "informal" description of the grammar:

$$S \to aSaSbSb \mid aSbSaSb \mid aSbSbSa \mid bSbSaSa \mid bSaSbSa \mid bSaSaSb \mid SS \mid \epsilon$$

6. Pick $s = a^p b^{p+1} c^{p+2}$. In the case analysis, using $|vxy| \leq p$, conclude that at least one symbol from $\{a, b, c\}$ is missing. Pump up or down according to different cases.

7. Note that a variable A is useful if and only if it satisfies the following conditions:

   (a) A must be "reachable" from the start symbol S, and

   (b) A must derive some string.

   To design an algorithm to find out all the useful variables in the grammar, use algorithms to find out the sets that satisfy (a) and (b) separately and take their intersection.

   We will compute the set $\mathcal{A}$ that satisfies (a) iteratively. Start with $\mathcal{A} = \{S\}$. Now for any rule $A \to w$, with $A \in \mathcal{A}$, add all the variables that appear in w to $\mathcal{A}$. Repeat till no new variable is added to $\mathcal{A}$.

   We again give an iterative algorithm to compute the set of variables $\mathcal{B}$ that satisfy (b). For any rule $B \to x$, where $x \in \Sigma^*$, add B to $\mathcal{B}$. After this initial step, for any rule of the form $A \to w$, where w is a string of terminals and variables from (the current) $\mathcal{B}$, add A to $\mathcal{B}$. Repeat till no new variables are added to $\mathcal{B}$.