Practice Problem Set #2 Solutions

Explanations have been intentionally kept verbose to remove as many doubts as possible on the eve of exams. Exam solutions should NEVER be this verbose.

1. $G = (\{S, A, B\}, \Sigma, P, S)$, where P consists of the following production rules:

$$\begin{aligned} S &\rightarrow 0S0 \mid 1S1 \mid A \\ A &\rightarrow 0B1 \mid 1B0 \\ B &\rightarrow 0B0 \mid 0B1 \mid 1B0 \mid 1B1 \mid \epsilon \end{aligned}$$

Intuitively, B generates all strings of 0s and 1s of even length (including ϵ). A generates all strings of even length (≥ 2) with different first and last symbols. S generates all strings of even length $2k$ ($k \geq 1$) such that for some $1 \leq l \leq k$, the i^{th} symbol from the beginning is different from the i^{th} symbol from the end.

Here's a proof that G serves our purpose. I don't expect you to give such proofs unless asked for. Nevertheless, you should have some reasoning like this at the back of your mind to convince yourself that a proposed grammar generates all strings of a given language and nothing more.

If $S \stackrel{i}{\Rightarrow} w$, it is easy to see that $i \geq 3$. For $i = 3$, only two derivations are possible: $S \Rightarrow A \Rightarrow 0B1 \Rightarrow 01$ and $S \Rightarrow A \Rightarrow 1B0 \Rightarrow 10$. Both strings satisfying our requirements. Let all terminal strings derivable from S in j steps ($j < i$) satisfy our requirement. Consider $S \stackrel{i}{\Rightarrow} w$, with $i > 3$. There are 3 cases:

- $S \Rightarrow 0S0 \stackrel{i-1}{\Rightarrow} 0y0 = w$, where $S \stackrel{i-1}{\Rightarrow} y$. By hypothesis, y is of the form y_1y_2 with $y_1 \neq y_2^R$ and $|y_1| = |y_2|$. Therefore, $0y0$ satisfies our requirements.
- $S \Rightarrow 1S1 \stackrel{i-1}{\Rightarrow} 1y1 = w$. Similar argument holds.
- $S \Rightarrow A \Rightarrow aBb \stackrel{i-2}{\Rightarrow} ayb = w$, where either $a = 0, b = 1$ or vice versa. In either case, the first and last symbols of w are different, so it can't be of the form xx^R. Since $B \stackrel{*}{\Rightarrow} y$, the number of symbols in y is even. Thus, w satisfies our requirements.

If w satisfies our requirements and $|w| = 2k$, there exists a smallest l $(1 \le l \le k)$ such that the k^{th} symbol of w from the left differs from the k^{th} symbol from the right. Thus, $w = x_1 a y_1 y_2 b x_1^R$, where x_1 is the substring composed of the first $l-1$ symbols ($x_1 = \epsilon$ if $l = 1$), a and b are the l^{th} symbols from the left and right ($a \ne b$), and y_1 and y_2 are composed of the $l+1^{st}$ to k^{th} symbols and $k+1^{st}$ to $2k-l^{th}$ symbols of w. Note $|y_1| = |y_2| = k - l$. It is now easy to see that $S \stackrel{l-1}{\Rightarrow} x_1 S x_1^R \Rightarrow x_1 A x_1^R \Rightarrow x_1 a B b x_1^R \stackrel{k-l}{\Rightarrow} x_1 a y_1 B y_2 b x_1^R \Rightarrow x_1 a y_1 y_2 b x_1^R = w$.

2. Suppose we didn't have parentheses in expressions. Then, a simple grammar for expressions would be:

$$\begin{aligned} S &\rightarrow A \circ S \mid A \\ A &\rightarrow \mathbf{id} \end{aligned}$$

Note that $S \rightarrow A \circ S$ forces generation of **id**s in an expression in a left-to-right order. Thus, the expression $\mathbf{id} \circ \mathbf{id} \circ \mathbf{id}$ can be generated in only one way as shown in Fig. 1. This also ensures that evaluating the tree results in a right-to-left evaluation of $\circ$, as desired.

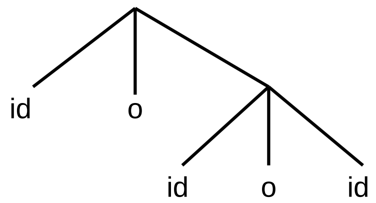

Figure 1:

With parentheses, each parenthesized subexpression (including the parentheses) behaves like an **id** when used as an operand of $\circ$. Thus, a straightforward extension of the previous grammar is:

$$\begin{aligned} S &\rightarrow A \circ S \mid A \\ A &\rightarrow \mathbf{id} \mid (S) \end{aligned}$$

To see intuitively why this grammar is unambiguous, consider an expression w generated by the grammar. If w contains no parentheses, by the argument for parentheses-free expressions, there is a unique derivation for w. If w contains parentheses, notice that there is only rule with parentheses on the right, i.e. $A \to (S)$. Moreover, this rule introduces matched pairs of parentheses enclosing whatever is subsequently derived from S. Consider a subexpression (e) of w such that e contains no parentheses, i.e. e is an innermost nested subexpression. From our argument for parentheses-free expressions, it follows that there is only one way to generate (e). The same applies to each innermost nested subexpression. This, in turn, implies that subexpressions at the next nesting level of parentheses can be generated in only one way. We can continue the argument until the outermost nesting levels are reached. Finally, there is one way of combining **id**s and subexpressions at the outermost level with $\circ$ to generate the desired expression.

3. The basic idea is to have two types of stack symbols, A_0 and A_1 (plus possibly an initial symbol Z_0). At any time, only one of these two types can be in the stack. If we have n A_0s in the stack, it means we have seen n more 0s than 1s so far. The interpretation for A_1s is similar. The acceptance criterion is to have a stack with no A_0s or A_1s.

   Thus, PDA $M = (\{q\}, \{0,1\}, \{Z_0, A_0, A_1\}, \delta, q, Z_0, \phi)$, where:

$$
\begin{aligned}
\delta(q, 0, Z_0) &= \{(q, A_0 Z_0)\} \\
\delta(q, 0, A_0) &= \{(q, A_0 A_0)\} \\
\delta(q, 0, A_1) &= \{(q, \epsilon)\} \\
\delta(q, 1, Z_0) &= \{(q, A_1 Z_0)\} \\
\delta(q, 1, A_0) &= \{(q, \epsilon)\} \\
\delta(q, 1, A_1) &= \{(q, A_1 A_1)\} \\
\delta(q, \epsilon, Z_0) &= \{(q, \epsilon)\}
\end{aligned}
$$

   Though you aren't required to prove the following unless specifically asked, it is easy to show by induction on the length of $|w|$ that

   (a) $(q, w, A_0) \overset{\star}{\vdash} (q, \epsilon, \epsilon)$ iff w has one more 1 than 0s.

   (b) $(q, w, A_1) \overset{\star}{\vdash} (q, \epsilon, \epsilon)$ iff w has one more 0 than 1s.

   (c) $(q, w, Z_0) \overset{\star}{\vdash} (q, \epsilon, \epsilon)$ iff w has equal no. of 1s and 0s.

   The proof is similar to that given in Example 4.3 of Hopcroft and Ullman.

To build the corresponding CFG, notice that M has a single state. So the technique used in class to show that for every CFG, there exists a PDA with a single state, can be applied in reverse. This yields $G = (\{Z_0, A_0, A_1\}, \{0, 1\}, P, Z_0)$ where P consists of:

$$\begin{aligned} Z_0 &\rightarrow 0A_0Z_0 \mid 1A_1Z_0 \mid \epsilon \\ A_0 &\rightarrow 0A_0A_0 \mid 1 \\ A_1 &\rightarrow 1A_1A_1 \mid 0 \end{aligned}$$

To see if G is ambiguous, we make the following observations:

- A sentential form in a leftmost derivation can have at most one instance of Z_0, and if present, it must be the rightmost variable. If you're not convinced, a simple induction on the number of steps in a derivation proves this.
- For every A_0 or A_1 in a sentential form, at least as many terminal symbols are generated in the final terminal string.

Now consider an arbitrary string $w \neq \epsilon$ generated by G. Let the i^{th} symbol be w_i. The grammar is ambiguous if we can have two distinct leftmost derivations for some w. Suppose at some point in a leftmost derivation, the next symbol to be generated is w_j. If the leftmost variable is A_0 or A_1, w_j uniquely determines the rule to apply next. If the leftmost variable is Z_0, this must be the only remaining variable since Z_0 is also the rightmost variable. Thus, if $Z_0 \rightarrow \epsilon$ is applied, the next symbol w_j cannot be generated. Thus, w_j once again uniquely determines the production rule to apply. In effect, the string w uniquely determines the rules by which each of its symbols are generated. When the entire string w is generated, the leftmost symbol must be Z_0 (A_0 or A_1 would generate at least one more symbol, so the string generated wouldn't be w). So, the only rule applicable now is $Z_0 \rightarrow \epsilon$. Finally, notice that $w = \epsilon$ also has a unique derivation $Z_0 \rightarrow \epsilon$.

4. Let $L_1 = L - \{\epsilon\}$. Since $a \in \Sigma$, $a \backslash L = a \backslash L_1$. Consider a CNF grammar G for L_1 and convert it to a PDA M with one state (recall how we proved equivalence of CFGs and PDAs in class). Let $M = (\{q_0\}, \Sigma, \Gamma, \delta, q_0, X_0, \phi)$. Each transition in δ has one of two forms:

- It consumes an input symbol and pops a symbol off the stack: $\delta(q_0, b, A) = \{(q_0, \epsilon)\}$ for $b \in \Sigma$, $A \in \Gamma$, or
- It doesn't consume any input and increases the symbols on the stack by one: $\delta(q_0, \epsilon, A) = \{(q_0, B_1C_1), \ldots (q_0, B_mC_m)\}$ for $m \geq 1$, $A, B_1, C_1 \ldots B_m, C_m \in \Gamma$.

Construct $M' = (\{q_0, q_1, q_2\}, \Sigma, \Gamma, \delta', q_1, X_0, \phi)$, where δ' is defined as follows:

- δ' contains all transitions of δ.
- For all $A \in \Gamma$, if $\delta(q_0, a, A) = \{(q_0, \epsilon)\}$, set $\delta'(q_1, \epsilon, A) = \{(q_0, \epsilon)\}$.
- For all $A, B, C \in \Gamma$, if $(q_0, BC) \in \delta(q_0, \epsilon, A)$, add (q_1, BC) to $\delta'(q_1, \epsilon, A)$.
- δ' contains no transitions other than those defined above.

$\underline{a\backslash L \subseteq N(M')}$: Suppose $aw \in L$, hence $\in N(M)$. It follows that $(q_0, aw, X_0) \overset{\star}{\vdash}_M (q_0, aw, A\gamma) \vdash_M (q_0, w, \gamma) \overset{\star}{\vdash}_M (q_0, \epsilon, \epsilon)$, for some $\gamma \in \Gamma^*$ and $A \in \Gamma$ such that $\delta(q_0, a, A) = \{(q_0, \epsilon)\}$. Since no transition in $(q_0, aw, X_0) \overset{\star}{\vdash}_M (q_0, aw, A\gamma)$ consumes any input, each is of the form $(q_0, \epsilon, B) \vdash_M (q_0, \epsilon, CD)$. By construction of δ', $(q_1, \epsilon, B) \vdash_{M'} (q_1, \epsilon, CD)$, so $(q_1, w, X_0) \overset{\star}{\vdash}_{M'} (q_1, w, A\gamma)$. The transition $(q_0, aw, A\gamma) \vdash_M (q_0, w, \gamma)$ implies $\delta(q_0, a, A) = \{(q_0, \epsilon)\}$. This in turn implies $\delta'(q_1, \epsilon, A) = \{(q_0, \epsilon)\}$. We therefore have $(q_1, w, X_0) \overset{\star}{\vdash}_{M'} (q_1, w, A\gamma) \vdash_{M'} (q_0, w, \gamma)$. Since all transitions of δ are also in δ', it follows that $(q_1, w, X_0) \overset{\star}{\vdash}_{M'} (q_0, w, \gamma) \overset{\star}{\vdash}_{M'} (q_0, \epsilon, \epsilon)$. Thus $w \in N(M')$.

$\underline{N(M') \subseteq a\backslash L}$: Suppose $w \in N(M')$. By construction, a transition that takes M' to state q_1 cannot pop a stack symbol or consume an input symbol. So, M' must eventually go to state q_0 to empty its stack, and until then no input is consumed. Also, once q_0 is entered, M' stays in q_0 since there is no transition from q_0 to q_1 in δ'. Thus, $(q_1, w, X_0) \overset{\star}{\vdash}_{M'} (q_1, w, A\gamma) \vdash_{M'} (q_0, w, \gamma) \vdash_{M'} (q_0, \epsilon, \epsilon)$, where $\gamma \in \Gamma^*$ and $A \in \Gamma$ such that $\delta'(q_1, \epsilon, A) = \{(q_0, \epsilon)\}$. But this implies $\delta(q_0, a, A) = \{(q_0, \epsilon)\}$ in PDA M. Since each transition in $(q_1, w, X_0) \overset{\star}{\vdash}_{M'} (q_1, w, A\gamma)$ takes M' to q_1, each is of the form $(q_1, \epsilon, A) \vdash_{M'} (q_1, \epsilon, BC)$. By construction, we then have $(q_0, \epsilon, A) \vdash_M (q_0, \epsilon, BC)$, so $(q_0, aw, X_0) \overset{\star}{\vdash}_M (q_0, aw, A\gamma) \vdash_M (q_0, w, \gamma)$. Since $(q_0, w, \gamma) \vdash_{M'} (q_0, \epsilon, \epsilon)$ and all transitions from q_0 in M' are also present in M, it follows that $(q_0, aw, X_0) \overset{\star}{\vdash}_M (q_0, w, \gamma) \overset{\star}{\vdash}_M (q_0, \epsilon, \epsilon)$. Thus $aw \in N(M)$ and hence in L.

5. (a) Not a CFL.

   If possible, let the language be a CFL, and let n be the Ogden's lemma (OL) constant. Consider $w = 0^n 1^{n+1} 2^{n+2}$ and consider the 0s as "marked". Let $w = uvxyz$ be the OL decomposition. Clearly, neither v nor y can contain more than one type of symbol; otherwise uv^2xy^2z would not be of the form $0^i1^j2^k$. Also, vy must contain at least one marked symbol, hence at least one 0. The following cases arise:

   - y contains only 0s. It follows that v is either ϵ or contains only 0s. Thus, uv^2xy^2z has at least as many 0s as 1s – a contradiction.
   - v contains only 0s. Then y is either ϵ or contains only 1s or only 2s or only 0s. The last case has been considered above. If y contains only 1s or is ϵ, uv^3xy^3z contains at least as many 0s as 2s – a contradiction. If y contains only 2s, uv^2xy^2z contains at least as many 0s as 1s – a contradiction.

   Thus, the language can't be a CFL.

   (b) The language can be expressed as $L_{01} \cup L_{02} \cup L_{12}$, where $L_{ij} = \{w \in (\mathbf{0}+\mathbf{1}+\mathbf{2})^* \mid w \text{ doesn't have the same number of } i\text{s and } j\text{s}\}$. We will show that each L_{ij} is a CFL, so the given language is a CFL too.

   We modify the PDA of Problem 3 slightly to construct a PDA that accepts L_{01} by final state. The formal construction is $M = (\{q, f\}, \{0, 1, 2\}, \{Z_0, A_0, A_1\}, \delta, q, Z_0, \{f\})$, where δ is defined by:

$$\begin{aligned}
\delta(q, 0, Z_0) &= \{(q, A_0Z_0)\} \\
\delta(q, 0, A_0) &= \{(q, A_0A_0)\} \\
\delta(q, 0, A_1) &= \{(q, \epsilon)\} \\
\delta(q, 1, Z_0) &= \{(q, A_1Z_0)\} \\
\delta(q, 1, A_0) &= \{(q, \epsilon)\}
\end{aligned}$$

$$\begin{array}{rcl}
\delta(q, 1, A_1) & = & \{(q, A_1A_1)\} \\
\delta(q, 2, Z_0) & = & \{(q, Z_0)\} \\
\delta(q, 2, A_0) & = & \{(q, A_0)\} \\
\delta(q, 2, A_1) & = & \{(q, A_1)\} \\
\delta(q, \epsilon, A_0) & = & \{(f, A_0)\} \\
\delta(q, \epsilon, A_1) & = & \{(f, A_1)\}
\end{array}$$

We know that the language accepted by the PDA of Problem 3 by empty stack is the set of strings containing an equal no. of 0s and 1s. It is then easy to argue that the language accepted by the above PDA by final state f is the set of strings containing unequal no. of 0s and 1s.

First, notice that consuming **2** has no effect on the state or stack – so **2**s may be ignored. The transition for popping Z_0, that has been eliminated, served the only purpose of emptying the stack of the final Z_0 in Problem 3. So eliminating this transition does not alter the sets of strings accepted by the PDA if we accept a Z_0 in the stack at the end. In other words, if at any stage, the stack contains only Z_0, the consumed input string contains an equal no. of 0s and 1s, otherwise not. The other two new transitions in PDA M take it to final state f. Since there is no transition out of f, these transitions can be executed only in the last step of a computation, when the entire input string has been read. Since these transitions can be executed if and only if A_0 or A_1 is on top of the stack, the final state can be entered if and only if there is an excess of 0s or 1s in the input string.

A similar argument shows that L_{12} and L_{02} are also CFLs.

(c) This is extremely simple. I will simply give the grammar and let you reason why it's correct. $G = (\{S, A\}, \{a, b\}, P, S)$ where P consists of:

$$\begin{array}{rcl}
S & \rightarrow & Ab \\
A & \rightarrow & aAa \mid aAb \mid bAa \mid bAb \mid a
\end{array}$$

(d) Not a CFL.

If possible, let the language be a CFL. Let n be the PL constant. Consider the string $w = 1^n01^n\#1^{n+1}0^n$ in the language. Let $w =$

$uvxyz$ be the PL decomposition, with $|vxy| \leq n$ and $|vy| \geq 1$. Clearly, neither v nor y can contain # as otherwise uv^2xy^2z will contain more than one #s. This means v and y, when not ϵ, must lie entirely on one side of #. The following cases arise:

- $y \neq \epsilon$ and y is to the left of # in w. Then uv^2xy^z increases the value of the number represented to the left of # without increasing the number to the right of # – a contradiction.
- $v \neq \epsilon$ and v is to the right of #. A similar argument leads to a contradiction.
- Either v or y is ϵ. Both can't be ϵ since $|vy| \geq 1$. Since v can only lie to the left of # and y to the right of #, if one of them is ϵ, both $uvxyz$ and uv^2xy^2z can't be in the language (either the left or the right number increases in uv^2xy^2z, but not both).
- Neither v nor y is ϵ, and v is to the left of # while y is to the right. Since $|vxy| \leq n$, it is easy to see that uxz is $1^n01^k\#1^l0^n$ where $l \leq n$ and $k \geq 1$. However, $1^n01^k + 1 = 1^{n+1}0^k$, so $1^n01^k\#1^l0^n$ can't be in the language – a contradiction.

6. $R - L$ need not be context-free. Let $R = (\mathbf{0} + \mathbf{1} + \mathbf{2})^*$ and L the CFL in Problem 5(b). Therefore, $R - L = L^c$. Now $L^c \cap \mathbf{0}^+\mathbf{1}^+\mathbf{2}^+ = \{0^n1^n2^n \mid n \geq 1\}$, a known non-CFL. So L^c must also be a non-CFL.

7. I'd like you to think more on this problem, so I'm going to give you a "not-so-good" solution. Here's an argument why you can always get a PDA that accepts L^R by the empty stack. While this argument gives a procedure for constructing the PDA, can you think of a more direct method for getting a PDA for L^R from a PDA for L?

   - Get a CNF grammar $G = (V, \Sigma, P, S)$ for $L - \{\epsilon\}$.
   - Construct a CNF grammar $G' = (V', \Sigma, P', S')$ for $L^R - \{\epsilon\}$ as follows:
     - $A' \in V'$ iff $A \in V$.
     - For each production rule $A \to BC$ in G, add the rule $A' \to C'B'$ in P'.
     - For each production rule $A \to a$ in G, add the rule $A' \to a$ in P'.

It is now easily shown that $A \stackrel{\star}{\Rightarrow} w$ in G iff $A' \stackrel{\star}{\Rightarrow} w^R$ in G' (use induction on the number of derivation steps).

- Finally, convert G' into a PDA $(Q, \Sigma, \Gamma, \delta, q_0, X_0, \phi)$ that accepts by empty stack.
- If $\epsilon \in L$, modify the PDA to $(Q \cup \{q_0'\}, \Sigma, \Gamma, \delta', q_0', X_0, \phi)$, where $q_0' \notin Q$ and δ' contains the transitions of δ plus the following:
  - $\delta(q_0', \epsilon, X_0) = \{(q_0', \epsilon), (q_0, X_0)\}$.

Think about how to get a PDA for L^R directly from one for L.