CS3133

HW#7

DUE: Thursday, October 7

1. (10 points) Show that the grammar

$$S \to 0S1 \mid 0S11 \mid \varepsilon$$

is ambiguous and show that the language it generates is not inherently ambiguous.

2. (18 points) For each of the following languages, tell whether or not it is context-free. Justify your answers.

a The set of binary strings that contain the substring 00 and the substring 11.

b $\left\{ zz^{rev} z \mid z \in \{0,1\}^* \right\}$, where z^{rev} is *z* written backwards. So

$$001111000011 \in \left\{ zz^{rev} z \mid z \in \{0,1\}^* \right\} .$$

c $\left\{ a^l b^m c^n \mid l + m = n \geq 0 \right\}$

3. (8 points) ***a*** Describe a PDA to accept $\left\{ 0^n \mid n \geq 0 \right\} \cup \left\{ 0^n 1^n \mid n \geq 0 \right\}$ by empty stack.

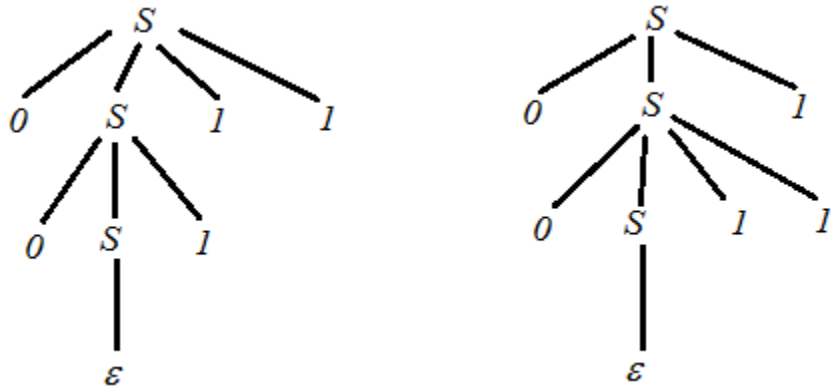
b Describe a PDA to accept $\left\{ a^i b^j c^k \mid (i, j, k \geq 0) \wedge (i = j \vee j = k) \right\}$ by final state.

4. ***a*** (6 points) Describe an algorithm that accepts as input a regular expression α and constructs a CFG *G* such that $L(G) = L(\alpha)$.

b (2 points) Justify why you can't do the opposite of part ***a***. That is, why can't you construct an algorithm that accepts as input an arbitrary CFG *G* and constructs a regular expression α such that $L(G) = L(\alpha)$?

CS3133
Solutions to HW#7

1. The grammar is ambiguous because string 00111 admits two distinct parse trees



The language is not inherently ambiguous because it is generated by the unambiguous grammar

$$S \to 0S1 | A$$
$$A \to 0A11 | \varepsilon$$

which starts by generating 0's with matching 1's on the outside (from *S*) and then, on the inside, generating 0's with matching pairs of 1's (from *A*). So, if $z = 0^n 1^{n+k}$, $0 \le k \le n$ then it is generated only by *n-k* applications of $S \to 0S1$, followed by one application of $S \to A$, followed *k* applications of $A \to 0A11$, followed by one application of $A \to \varepsilon$.

2. ***a*** It is a CFL because it is generated by the CFG

$$S \to A_{00..11} | B_{11..00}$$
$$A_{00..11} \to C_{0s1s}00C_{0s1s}11C_{0s1s}$$
$$B_{11..00} \to C_{0s1s}11C_{0s1s}00C_{0s1s}$$
$$C_{0s1s} \to 0C_{0s1s} | 1C_{0s1s} | \varepsilon$$

b $\left\{ zz^{rev}z \mid z \in \{0,1\}^* \right\}$ is not context-free. Assume it is a CFL, and let *k* be as provided by the PL for CFLs. Choose $z = 0^k 1$. We know that $0^k 110^{2k} 1 \in \left\{ zz^{rev}z \mid z \in \{0,1\}^* \right\}$. The PL assures us that $0^k 110^{2k} 1$ can be written as *uvwxy* with $|vx| \ge 1$ and $|vwx| \le k$. We note that *v* and *x* can not contain 1's from both the first string 11 and the final 1. There are two cases to consider:

Case 1 Exactly one of *v* or *x* contains a 1: In this case *uwy* drops at least one 1 from 11 or from the final 1, and in this case it can't belong to $\left\{ zz^{rev}z \mid z \in \{0,1\}^* \right\}$, which contradicts the Pumping Lemma.

Case 2 String vx must span a nonempty string of 0's either from the first string of k 0's or the second string of $2k$ 0's. In either case, uv^2wx^2y will add 0's to exactly one of 0^k or 0^{2k}, which can't belong to $\left\{zz^{rev}z \mid z \in \{0,1\}^*\right\}$, contradicting the Pumping Lemma.

So $\left\{zz^{rev}z \mid z \in \{0,1\}^*\right\}$ can not be a CFL.

c It is a CFL because it is generated by the CFG

$$S \to aSc \mid A$$
$$A \to bAc \mid \varepsilon$$

3. ***a*** One idea is to guess whether the string belongs to $\{0^n \mid n \geq 0\}$ (and go to q_{0^n}) or to $\{0^n1^n \mid n \geq 0\}$ (and go to $q_{0^n1^n}$). So $\{0^n \mid n \geq 0\} \cup \{0^n1^n \mid n \geq 0\}$ is accepted by empty stack by PDA $\left(\left\{s, q_{0^n}, q_{0^n1^n}, q^*_{0^n1^n}, q_{final}\right\}, \{0,1\}, \{\perp\}, s, \perp, \{q_{final}\}\right)$ where

$$\delta(s, \varepsilon, \perp) = \left\{\left(q_{0^n}, \varepsilon\right), \left(q_{0^n1^n}, \varepsilon\right)\right\},$$
$$\delta\left(q_{0^n}, \varepsilon, \varepsilon\right) = \left\{\left(q_{final}, \varepsilon\right)\right\},$$
$$\delta\left(q_{0^n}, 0, \varepsilon\right) = \left\{\left(q_{o^n}, \varepsilon\right), \left(q_{final}, \varepsilon\right)\right\},$$
$$\delta\left(q_{0^n1^n}, 0, \varepsilon\right) = \left\{\left(q_{0^n1^n}, 0\right)\right\}$$
$$\delta\left(q_{0^n1^n}, 1, 0\right) = \left\{\left(q^*_{0^n1^n}, \varepsilon\right)\right\}$$

b One idea is to guess whether the input string belongs to $\{a^i b^i c^k \mid i,k \geq 0\}$ or to $\{a^i b^k c^k \mid i,k \geq 0\}$ (by going to states $q_{a=b}$ or $q_{b=c}$ respectively). The only final state is q_{final} and the transition function is:

$$\delta(s, \varepsilon, \perp) = \left\{(q_{a=b}, \perp), (q_{b=c}, \perp)\right\}$$
$$\delta(q_{a=b}, a, \perp) = \delta(q_{a=b}, a, a) = \left\{(q_{a=b}, aa)\right\}$$
$$\delta(q_{a=b}, b, a) = \delta\left(q^*_{a=b}, b, a\right) = \left\{\left(q^*_{a=b}, \varepsilon\right)\right\}$$
$$\delta\left(q^*_{a=b}, \varepsilon, \perp\right) = \left\{(q_{consume-c}, \varepsilon)\right\}$$
$$\delta(q_{consume-c}, c, \perp) = \left\{(q_{consume-c}, \perp)\right\}$$
$$\delta\left(q_{consume-c}, \varepsilon, \perp\right) = \left\{\left(q_{final}, \perp\right)\right\}$$
$$\delta(q_{b=c}, a, \perp) = \left\{(q_{b=c}, \perp)\right\}$$
$$\delta(q_{b=c}, \varepsilon, \perp) = \left\{\left(q^*_{b=c}, \perp\right), \left(q_{final}, \perp\right)\right\}$$
$$\delta(q_{b=c}, a, \perp) = \left\{(q_{b=c}, \perp)\right\}$$
$$\delta\left(q^*_{b=c}, b, \perp\right) = \left\{\left(q^*_{b=c}, b \perp\right)\right\}$$
$$\delta\left(q^*_{b=c}, b, b\right) = \left\{\left(q^*_{b=c}, bb\right)\right\}$$

$$\delta\left(q^{*}_{b=c}, c, b\right) = \delta\left(q^{**}_{b=c}, c, b\right) = \left\{\left(q^{**}_{b=c}, \varepsilon\right)\right\}$$
$$\delta\left(q^{**}_{b=c}, \varepsilon, \perp\right) = \left\{\left(q_{final}, \varepsilon\right)\right\}$$

4. ***a*** We convert α by recursively breaking it down.

The three base cases are:

if $\alpha = \varnothing$ **then return** $G = \left(\{S\}, \Sigma, \varnothing, S\right)$

if $\alpha = \varepsilon$ **then return** $G = \left(\{S\}, \Sigma, \{S \to \varepsilon\}, S\right)$

if $\alpha = a$ **then return** $G = \left(\{S\}, \Sigma, \{S \to a\}, S\right)$

We can now simply invoke closure of regular languages under union, concatenation and Kleene star. That is, if $\alpha = \beta + \gamma$, then we compute the grammars to generate $L\left(\beta\right)$ and $L\left(\gamma\right)$, and then construct the grammar to generate $L\left(\beta\right) \cup L\left(\gamma\right)$. If $\alpha = \beta\gamma$, then we compute the grammars to generate $L\left(\beta\right)$ and $L\left(\gamma\right)$, and then construct the grammar to generate $L\left(\beta\right)L\left(\gamma\right)$. If $\alpha = \beta^{*}$, then we compute a grammar to generate $L\left(\beta\right)$, and then construct the grammar to generate $L\left(\beta^{*}\right)$.

b If an algorithm existed to accept as input any CFG and produce a regular expression to describe the same language, then when it accepted as input the grammar with productions $S \to 0S1|\varepsilon$, it would produce a regular expression describing $\left\{0^{n}1^{n} \,|\, n \geq 0\right\}$. But this is impossible, since $\left\{0^{n}1^{n} \,|\, n \geq 0\right\}$ is not a regular language. So, by contradiction, the algorithm does not exist.