ComS 331 Spring 2024 Name: Aren Ashlock

1. A *fully parenthesized arithmetic expression* on the natural numbers is defined recursively as:

   (a) A natural number expressed in decimal notation (e.g., 0, 4, 357, but not 007).

   (b) $(\alpha + \beta)$ or $(\alpha * \beta)$, where α and β are fully parenthesized arithmetic expressions.

   Define a CFG that generates the following language over $\{0, 1, ..., 9, (,), *, +, o, d, e, v, n\}$:

   $$L = \{\alpha \ odd : \alpha \text{ is a fully parenthesized arithmetic expression with an odd value}\}$$
   $$\cup \{\alpha \ even : \alpha \text{ is a fully parenthesized arithmetic expression with an even value}\}$$

   For example, "75 *odd*", "((241 * 2) + 40034) *even*" are strings in the language (ignore the blank spaces in the strings, they are just there for clarity).

   $G = (\{S, A, B, C, D, E, F\}, \{0, 1, ..., 9, (,), *, +, o, d, e, v, n\}, S, \{S \to Aeven|Bodd,$
   $A \to 0|2C|4C|6C|8C|1D|3D|5D|7D|9D|(A * A)|(A * B)|(B * A)|(A + A)|(B + B),$
   $B \to 1E|3E|5E|7E|9E|2F|4F|6F|8F|(B * B)|(A + B)|(B + A),$
   $C \to \epsilon|0C|2C|4C|6C|8C|1D|3D|5D|7D|9D,$
   $D \to 0C|2C|4C|6C|8C|1D|3D|5D|7D|9D,$
   $E \to \epsilon|1E|3E|5E|7E|9E|0F|2F|4F|6F|8F,$
   $F \to 1E|3E|5E|7E|9E|0F|2F|4F|6F|8F\})$

2. Define a context-free grammar for the language $L = \{0^n 1^m 0^m 1^n : n, m \in \mathbb{N}\}$.

   $G = (\{S, A\}, \{0, 1\}, S, \{S \to 0S1|A|\epsilon, A \to 1A0|\epsilon\})$

3. Define a context-free grammar for the language $L = \{uv \in \{0, 1\}^* : |u| = |v| \wedge u \neq v^R\}$.

   $G = (\{S, A\}, \{0, 1\}, S, \{S \to 0A1|1A0|1S1|0S0, A \to 0A0|0A1|1A0|1A1|\epsilon\})$

4. Let $|x|_a$ be the number of occurrences of the symbol a in the string x.

   - Define a context-free grammar for the language $L = \{w \in \{0, 1\}^* : |w|_0 = |w|_1\}$.
   - Give a formal proof, or at least the key idea(s), of why your grammar generates L.

   $G = (\{S, A, B\}, \{0, 1\}, S, \{S \to 0AS|1BS|\epsilon, A \to 1|0AA, B \to 0|1BB\})$

   Proof:

   My grammar generates the language from left to right. While in state S, which is essentially the "balanced" # of 1s and 0s, if you add a 1 or 0, it adds a variable that forces the opposite number to be added before you can reach state S again. The key is that state S is the only state in which all variables can be removed leaving only terminals for the language to accept.

   As for states A and B, these states have 2 choices. You can either choose the number that causes the string to get closer to being balanced, or you can add another of the number that appears more than the other number. For the latter option, it adds another variable signifying that another of the less-appearing number is required to reach a "balanced" state. The choice is crucial since we aren't requiring a certain pattern, but rather that the # of 1s and 0s are "balanced." Until all those variables are removed, state S cannot be variable-free, thus the language will not accept the string.