CS 181 Homework 7

Charles Zhang, 305-413-659

May 15, 2021

Problem 1

$$\Sigma = \{a, b, c, d\}$$

$$S \longrightarrow SS \mid P \mid B \mid \epsilon$$

$$P \longrightarrow aSc \mid ac$$

$$B \longrightarrow bSd \mid bd$$

a) Proof (non-intensive):

- This CFG can be shown to be ambiguous, as the empty string can be derived using multiple leftmost derivations
- $S \longrightarrow SS \longrightarrow \epsilon S \longrightarrow \epsilon$
- $S \longrightarrow \epsilon$

b) Justification:
Due to the rule $S \longrightarrow SS \mid P \mid B \mid \epsilon$, we know that there are infinitely many ways to generate any number of P, B, or ϵ by combining $S \longrightarrow SS$ and $S \longrightarrow \epsilon$. Since we can generate a sequence of non-terminals in infinitely many ways, we know that the resultant strings derived from those terminals can also be generated in infinitely many ways, making the grammar ambiguous.

c) CFG: $G = \{V, \Sigma, R, S\}$ where $V = \{S, X, P, B\}$ and R is the rule set below:

$$S \longrightarrow \epsilon \mid X$$

$$X \longrightarrow XP \mid XB \mid P \mid B$$

$$P \longrightarrow aXc \mid ac$$

$$B \longrightarrow bXd \mid bd$$

Justification:
We handle the issue of being able to use $S \longrightarrow SS$ and $S \longrightarrow \epsilon$ by splitting the empty string case into its own rule. Once this occurs, we know there will be at least one of P or B, so we use tail-recursion in the X rule to ensure that there is one unique way to generate a given number/ordering of Ps and Bs. From there, we can simply use the same rules for P and B to make sure our unambiguous CFG recognizes the same language as the ambiguous one.

Problem 2

a) No
Justification:
C_0 doesn't correctly represent the language. In attempting to find a cycle containing $NODE0$, the TM takes the first outgoing node and follows it. However, it is possible that the outgoing edge that is selected will not be part of a cycle, even if $NODE0$ is part of a cycle, since there could be multiple outgoing edges from $NODE0$. Since the algorithm contains no mechanism to backtrack, this would lead to an improper rejection of the language.

b) This is a procedure. If during computation, the TM were to find a cycle that doesn't contain $NODE0$, it is possible that the TM will continue to traverse this cycle. Since the while loop in the pseudocode only terminates if specifically $NODE0$ is revisited, a cycle that doesn't contain $NODE0$ results in a computation that never halts.

Problem 3

$$\Sigma = \{a, b\}$$

$$L = \left\{ w \in \Sigma^+ \;\middle|\; |w| \text{ is even, and } w \text{ contains at least one } a \text{ in the first half} \right.$$
$$\left. \text{and exactly one } b \text{ in the second half} \right\}$$

CFG: $G = \{V, \Sigma, R, S\}$ where $V = \{S, A, B, F\}$ and R is the rule set below:

$$S \longrightarrow aFb \mid aAa \mid bBb \mid bSa$$

$$A \longrightarrow aFb \mid aAa \mid bFb \mid bAa$$
$$B \longrightarrow aFa \mid bBa$$
$$F \longrightarrow aFa \mid bFa \mid \epsilon$$

Justification: This CFG accomplishes the specified language by using the idea that, as we build a string from the inside out, terminals on the left side will end up in the first half of the string and terminals on the right side will end up in the second half of the string. From there, the CFG splits strings up into 4 categories: strings where at least one a has been placed on the left (A), strings where one b has been placed on the right (B), strings that have both at least an a on the left and one b on the right (F), and strings that have satisfied neither constraint (S). When terminals are added to each side, the CFG sends the derivation into the appropriate rule, ensuring that the constraint cannot be broken later on. The CFG can only accept from the F rule, where all constraints have been satisfied. Furthermore, building the string from the inside out makes it possible to ensure the string is of even length.