CS 181: Formal Languages and Automata Theory **Spring 2021**

Homework 7 Solution

Assigned: 11 May *Due: 17 May 9:00pm PDT*

Note Submission deadline is 9:00pm.

Problem 1

Let $\Sigma = \{a, b, c, d\}$. Consider the following CFG. The start symbol is S:

$$\begin{aligned} S &\longrightarrow SS \mid P \mid B \mid \epsilon \\ P &\longrightarrow aSc \mid ac \\ B &\longrightarrow bSd \mid bd \end{aligned}$$

a. Prove that this grammar is ambiguous.

b. Briefly explain how the grammar works in a manner that causes it to be ambiguous.

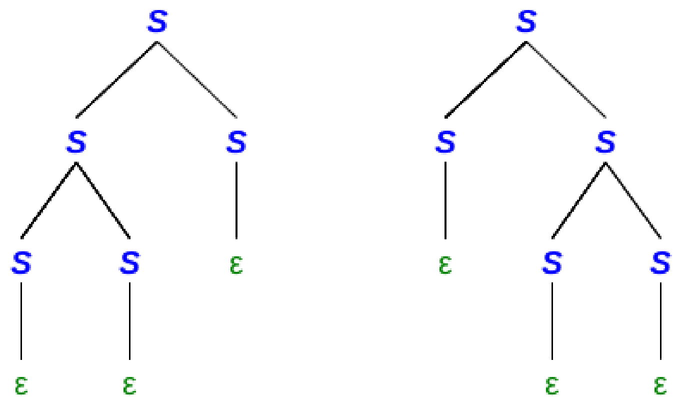
c. Define a new grammar that is unambiguous or "fix" the given grammar to make it unambiguous. As always, *briefly* explain how your grammar is designed to be correct and unambiguous.

Soutions:

a. To "prove" that a CFG is ambiguous, we only need to show that the grammar generates one string ambiguously by showing two different left-most derivations, two different right-most-derivations, or two different parse trees. We show two different left-most derivations for the string ϵ:

$$\underline{S} => \epsilon$$
$$\underline{S} => \underline{S}S => \epsilon\underline{S} => \epsilon$$

b. The grammar contains two sources of ambiguity. (You only needed to identify one for full credit.) First: as illustrated by the previous answer, it can generate any number of variable S which can then be replaced by ϵ. So the string ϵ can be generated by infinitely many left-most derivations. Also, any non-empty string can be generated by multiple left-most derivations by using the first rule for S to introduce unnecessary S's and then "erasing" them with the last rule for S. Second, the rule "$S \longrightarrow SS$" is a source of ambiguity in another way. For any string with a derivation that has at least three S's, this grammar can lead to multiple left-most derivations (and equivalently, multiple parse trees) which associate the three S's in different ways. This case is illustrated by the two parse trees in Figure 1 below. We used "ϵ" here again just for convenience, but we could have replaced all the S's with something like "ac" via variable P, and the same ambiguity would appear.



(a) (b)

Figure 1: (a) and (b) are two distinct parse trees for the string ϵ

c. We can define a new unambiguous grammar as follows (also with start variable S):

$$\begin{aligned} S &\longrightarrow T \mid \epsilon \\ T &\longrightarrow BT \mid PT \mid B \mid P \\ P &\longrightarrow aTc \mid ac \\ B &\longrightarrow bTd \mid bd \end{aligned}$$

This grammar is unambiguous. There is only one way to generate the ϵ string, since variable T will always generate at least one B or P. Variables B and P cannot generate the ϵ string and can only generate a pair of bd or ac respectively. The production rules of variable T are used to generate the sequence of B's and P's, and there is only one parse tree for every B, P sequence. Between each ac or bd pair, the process begins again unambiguously with variable T.

Problem 2

Let Σ be some alphabet appropriate for representing digraphs. Assume that the encoding represents the graph as a list of edges in the form of ordered pairs of nodes represented in some appropriate manner. (Assume the set of all nodes of the graph can be determined from the list of edges.) The pseudo-code below defines a Deterministic TM, $C0$, which is an attempt to program a TM to recognize the following language, L, over Σ: L equals all strings which encode a graph, G, such that G has at least one directed cycle containing the first node in the encoding of the graph as a string. The first node in the graph encoding is defined as the source node in the first ordered pair in the list of edges - we call this node "$NODE0$". Answer the following questions about the pseudo-code for $C0$:

a. Does $C0$ correctly recognize the language? Briefly explain why or why not.

b. Regardless of the correctness of the pseudo-code, Is $C0$ an algorithm? or is it a procedure?

Pseudo-code for $C0$:
$C0$ uses a work tape to store a value, "CurrentNode", used during the computation (along with other values used to implement the details of the steps which we are assuming can be done based on Church's Thesis).

```
Initialization:
Test whether the input tape contains a valid encoding of a digraph.
If not, halt and reject.
If there is at least one edge out-going from "NODE0", then choose the first outgoing edge
from "NODE0", and call that edge (NODE0, NextNode),
else halt and reject.
Write the value "NextNode" to the location "CurrentNode" on the work tape.

Begin While the value of "CurrentNode" on the work tape does not equal $NODE0$:

1. If there is at least one edge out-going from "CurrentNode", then choose the first outgoing
   edge from "CurrentNode", and call that edge (CurrentNode,NextNode). Write the value
   NextNode to the location $CurrentNode$ on the work tape

   else halt and reject.

End while loop.
Halt and accept.
```

Solutions:

a. No. This process is similar to depth-first search but without checking to see if it has reached a node for the second time. If the graph contains a loop that does not pass through the first node, then it is possible that the process will get "caught" in that loop and never consider other parts of the graph which might contain a loop that does include the first node.

b. For the same reason, $C0$ is not an algorithm. It can go into an infinite loop on some inputs regardless of whether it should have accepted that input according to the definition of the intended language. Since it does not halt for all possible inputs, it is a *procedure* and not an *algorithm*.

Problem 3

Let $\Sigma = \{a, b\}$, and let L be the language over Σ:

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

Give a context-free grammar (CFG) for L, and briefly explain how it works to correctly represent the language. There are no additional requirements for your solution other than what is stated here, except the usual guidance to make sure your answer is clear and understandable and avoids too much unnecessary complexity. In particular, there is no credit for the grammar being unambiguous (if possible.)

Solution:

We define the CFG with start variable S as follows:

$$\begin{aligned}
S &\longrightarrow bSa \mid aAa \mid bCb \mid aBb \mid ab \\
A &\longrightarrow Tb \mid TAa \mid TBb \\
B &\longrightarrow Ta \mid TBa \\
C &\longrightarrow aa \mid aBa \mid bCa \\
T &\longrightarrow a \mid b
\end{aligned}$$

where the start variable is S.

Brief explanation:

- The variable A generates all possible even-length strings whose second half has exactly one b. Thereby, all strings generated from the production rule $S \longrightarrow aAa$ are in the given language L.
- The variable B generates all possible even-length strings whose second half does not have any b. Thereby, all strings generated from the production rule $S \longrightarrow aBb$ are in the given language L.
- The variable C generates all possible even-length strings whose first half has at least one a's and whose second half does not have any b. Thereby, all strings generated from the production rule $S \longrightarrow bCb$ are in the given language L.
- Taken together, the variable S generates all possible even-length strings whose first half has at least one a and whose second half has exactly one b.