CS375 Homework Assignment 7 Solution Set (40 points)
Due date: November 13, 2024

1. (4 points)
The language generated by the following grammar is (1 point)

$\{aba^n, abca^m \mid n, m \in N\}$

S →abA | abcA A → aA | Λ

This grammar is LL(3). (1 point)
Use left-factoring we can find an equivalent LL(k) grammar for this grammar where k is as small as possible. In the following, fill out the blank in the middle portion to make the resulting grammar such an LL(k) grammar.

S → abT T → A | cA A → aA | Λ (1 point)

What is the value of k? k = 1 (1 point)

2. (4 points)
Find an equivalent grammar with no left recursion for the given grammar:
S → SaaS | ab.

S → abA A → aaSA | Λ
S → abA A → aaabAA | Λ
or
S → abA A → aaabA | Λ
(either one is okay)

(2 points)

Is the resulting grammar LL(k)? x Yes ☐ No (1 point for Yes, 2 points for No)

If your answer is YES, then what is the value of k? k = 1 (1 point)

3. (10 points)
The following CFG is an LL(2) but not an LL(1) grammar for the language $L =$

$\{a^n b \mid n \in N\}$

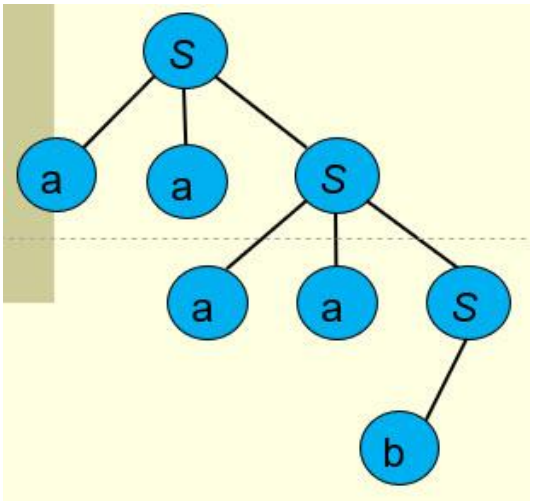
S → aaS | ab | b (∗)

In the following blanks, using this grammar, show a unique left-most derivation for the string $aaaab$ by scanning two symbols of the string each time and then fill out the empty nodes in the tree on the righthand side to make it a parse tree for the string.

$S \Rightarrow$ aaS

$S \Rightarrow$ aaaaS

$S \Rightarrow$ aaaab (1.5 points)

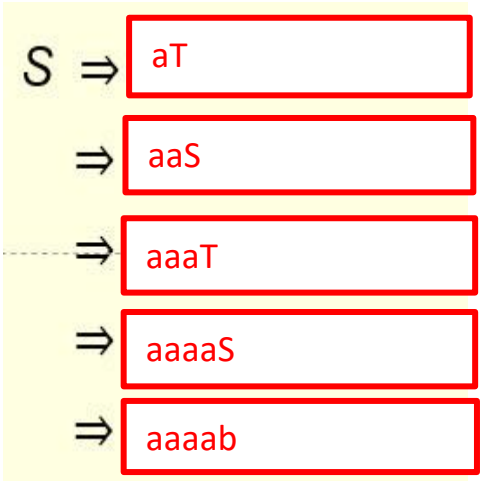


(2 points)

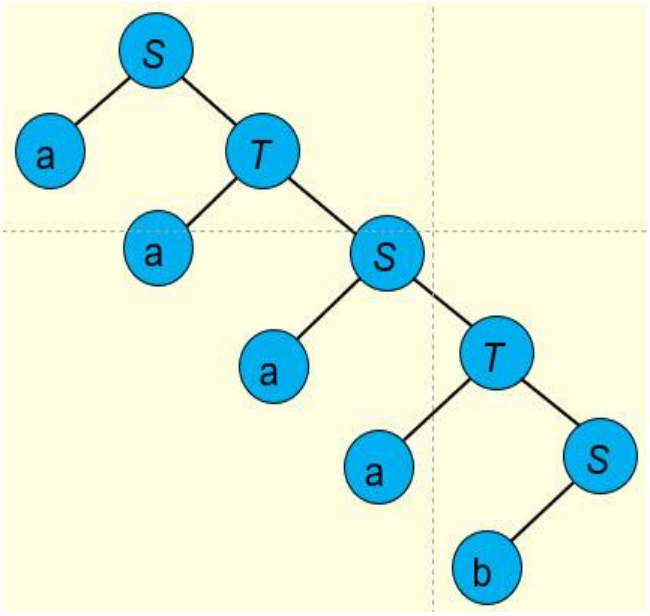
Using "*left-factoring*" technique we can convert (∗) to a CFG of the following form

S → aT | b T → aS | b (∗∗)

This new CFG is an LL(1) grammar for the language *L*. In the following blanks, using grammar (∗∗), show a unique left-most derivation of the string $aaaab$ by scanning only one symbol of the string each time and then fill out the empty nodes in the tree underneath to make it a parse tree for this string with respect to the new CFG (∗∗).



(2.5 points)



(4 points)

There is a special relationship between this parse tree and the parse tree you get for (*). Namely, if you merge each T node in the above tree with its parent node (an S node), you get the parse tree for (*). Note that in the notes "Context-free Languages and Pushdown Automata-IV", it is stated that *if the old grammar is not ambiguous then the new grammar is not ambiguous either*. This special relationship tells you why this property is true.

4. (5 points)

The following CFG is a grammar for the language $L = \{a^n,\ a^n b^m \mid n \geq 1, m \geq 2\}$

S → aS | abB | a B → bB | b $(*)$

It is not LL(1). Is this CFG an LL(2) grammar for the language *L*? Mark the box you think

is the right answer below:

YES [ ] NO [x] (1 point)

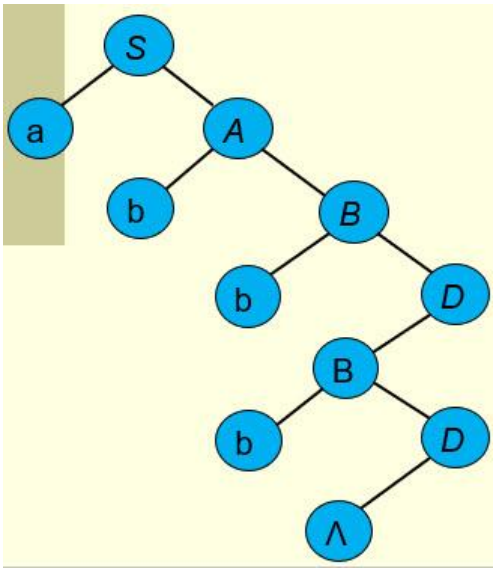
(consider the string *aabb* see if you can determine the unique production(s) required for each step by scanning only two symbols of the input string each time).

We can use the "*left-factoring*" technique to convert $(*)$ to a CFG of the following form

$S \rightarrow aA$ $\quad B \rightarrow bD$

$A \rightarrow S \mid bB \mid \Lambda$ $\quad D \rightarrow B \mid \Lambda$ $\quad (**)$

In the following, fill out the empty nodes in the tree to make it a parse tree for the string abbb with respect to the new CFG, $(**)$, by scanning only one symbol of the string each time to make a decision on which production(s) to use for the next step in the parsing process. (Is the new CFG $(**)$ an LL(1) grammar?)



(4 points)

5. (6 points; 1 point each blank)

The following given grammar is a left recursive grammar

$S \rightarrow Sbab \mid b \mid a$

The language generated by this grammar is of the following form:

L = { bw, aw | w = [$(bab)^n, \quad n \in N$] }

This left recursive grammar can be transformed to a right recursive grammar as follows:

S → bB | aB

B → babB | Λ

This right recursive grammar is an LL(1) grammar.

6. (6 points; 1 point each blank)
   The following grammar is an indirect left recursive grammar

   S → Ba | b B → Sb | a

   Strings of the language generated by this grammar are of the following form:

   L = { bw, aaw | w = $(ba)^n, \quad n \in N$ }

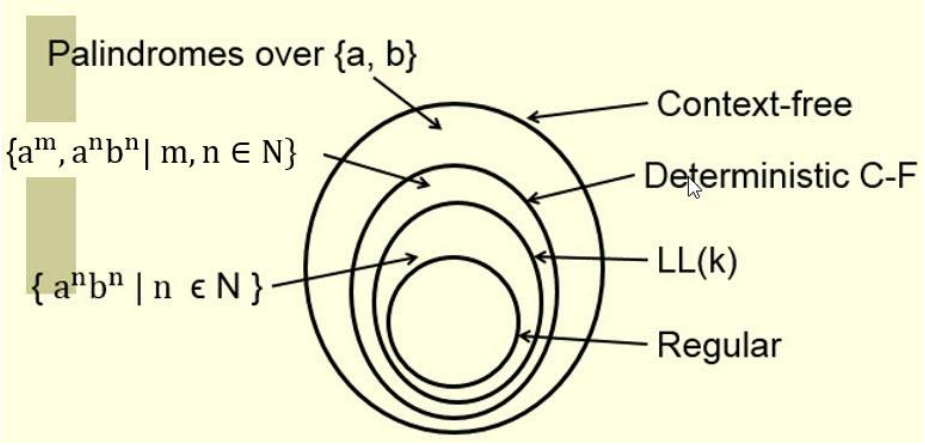
   This indirect left recursive grammar can be transformed to a right recursive grammar as follows:

   S → aaB | bB

   B → baB | Λ

   This right recursive grammar is an LL(1) grammar.

7. (5 points)
   Slide 45 of the notes "Context-free Languages and Pushdown Automata IV" shows that the set of LL(k) languages is a proper subset of the set of deterministic C-F languages (or see the following figure). In particular, it points out that the language $\{a^m, a^n b^n \mid m, n \in N\}$ is a deterministic C-F, but not LL(k) for any k.



To show the language is not LL(k) for any k, note that the grammar for this language is

$S \to A \mid B \quad A \to aA \mid \Lambda \quad B \to aBb \mid \Lambda$

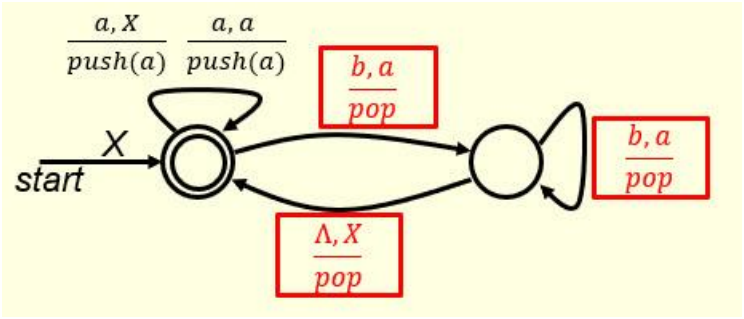
or

$S \to A \mid B \quad A \to aAb \mid \Lambda \quad B \to aB \mid \Lambda$ (2 points)

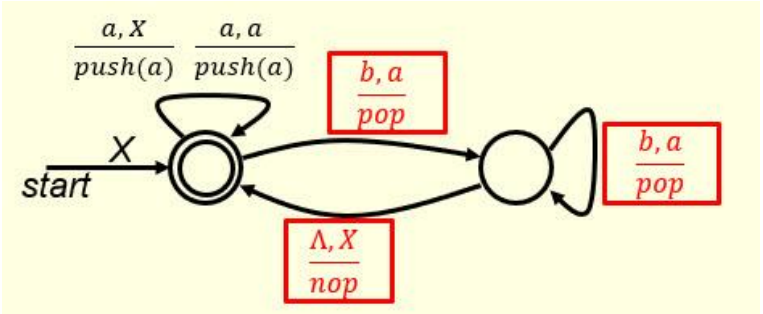
(you only need to answer one case here, either one). The language contains Λ as an element. Now consider the case k = 1 and consider the input string ab. When the first symbol is scanned, we get an 'a'. This information alone is not enough for us to make a proper choice. So we don't even know what to do with the first step in the parsing process.

For k = 2, if we consider the input string aabb, we face the same problem. For any k > 2, the input string $a^k b^k$ would cause exactly the same problem. So this grammar is not LL(k) for any k.

On the other hand, by putting proper instructions into the blanks of the following figure, we get a deterministic final-state PDA that accepts the language $\{a^m, a^n b^n \mid m, n \in N\}$.



or



(3 points)

(again, you only need to answer one case here, either one). Hence, this language is indeed deterministic C-F, but not LL(k) for any k.