CS375 Homework Assignment 7 （40 points）

Due Date: Nov. 13, 2026

1. (6 points)

The following CFG is an LL(3) grammar for the language $L = \{a^n b \mid n \in N\}$

S → aaaS | aab | ab | b (∗)

But it is not LL(2). We can use the "*left-factoring*" technique to convert (∗) to a CFG of the following form

S → aR | b R → aaS | ab | b (∗∗)

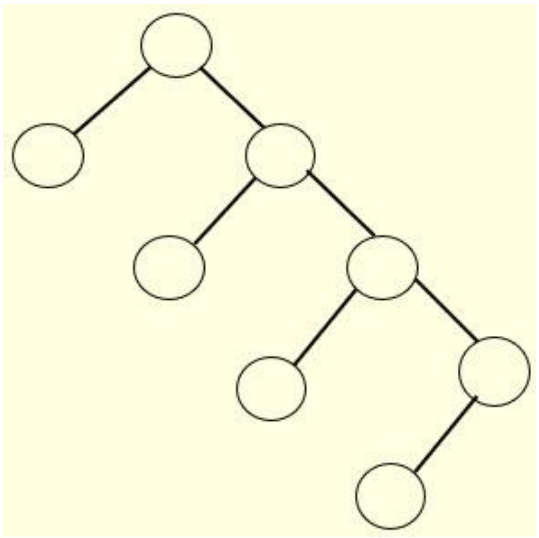
Is this new CFG an LL(2) grammar for the language *L*? Mark the box you think is the right answer below:

YES ☐ NO ☐ (1 point)

If we apply the "*left-factoring*" technique on R in (∗∗), we can convert (∗∗) into the following form

S → aR | b R → aT | b T → aS | b (∗∗∗)

This is also a CFG for the language *L*. In the following, fill out the empty nodes in the following tree to make it a parse tree for the string aaab with respect to the new CFG, (∗∗), by scanning only one symbol of the string each time to make a decision on which production to use for the next step in the parsing process.



(4 points)

In general, is the CFG, (∗∗∗), an LL(1) grammar for the language *L*?

YES ☐ NO ☐ (1 point)

2. (7 points)

The following CFG is an LL(2) 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 $(*)$

But it is not LL(1). We can use the "*left-factoring*" technique to convert $(*)$ to a CFG of the following form

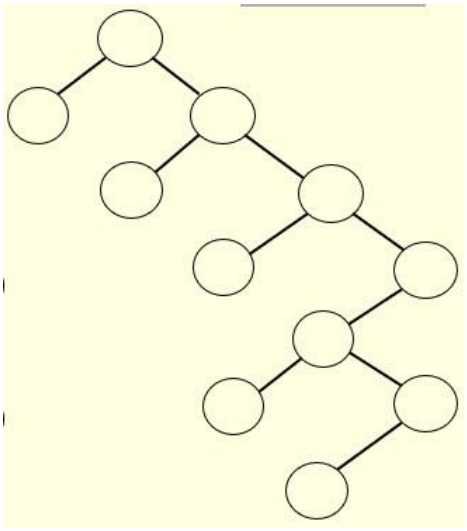
S → aA B → bD

A → S | bB | Λ D → B | Λ $(**)$

Is this new CFG an LL(1) grammar for the language *L*? Mark the box you think is the right answer below:

YES ☐ NO ☐ (1 point)

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.



(6 points)

3. (8 points)

Use left-factoring one can find an equivalent LL(k) grammar for the following grammar with k being as small as possible. In this case, left-factoring will create a production that is always used as the first step in the left-most derivation and the common factor contained on the right side of that production will provide an automatic match on the first part of the input string.

S → abA | abcS A → aA | Λ

The language generated by the given grammar is:

☐ (2 points)

The given grammar is LL(3).
By factoring ab out from S → abA | abcS, the given grammar can be converted to

S → abT T → A | cS A →aA | Λ (1)

This grammar can also be written as

S → abT T → aA | cS | Λ A → aA | Λ (2)

Grammars (1) and (2) are both LL(1). To show (2) is LL(1), it is sufficient to consider ab and abcaba as input strings. In the following, show a unique leftmost derivation of the string abcaba using grammar (2).

S ⇒ [ ]
⇒ [ ]
⇒ [ ]
⇒ [ ]
⇒ [ ]
= [ ] (6 points)

4. (6 points; 1 point each blank)
The following given grammar is a left recursive grammar

S → Sbab | b | a

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

L = { bw, aw | w = [ ] }

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

S → [ ] | [ ]

B → [ ] | [ ]

This right recursive grammar is an LL([ ]) grammar.

5. (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 = [ ] }

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

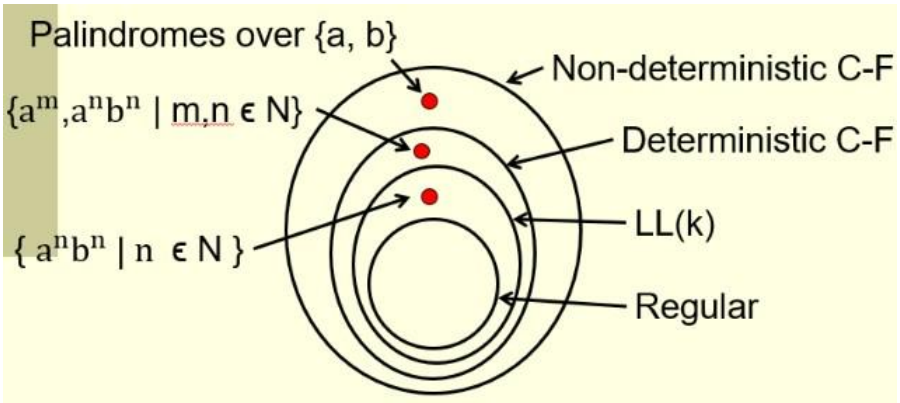
S → [ ] | [ ]

B → [ ] | [ ]

This right recursive grammar is an LL([ ]) grammar.

6. (7 points)

In slide 56 of the notes "Context-free Languages and Pushdown Automata IV", it is shown 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 deterministic C-F, but not LL(k) for any k.



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

S → A | B A → [ ] | [ ] B → [ ] | [ ]

or

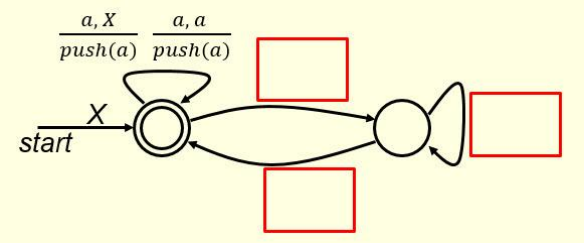
S → A | B A → [ ] | [ ] B → [ ] | [ ]

(4 points)

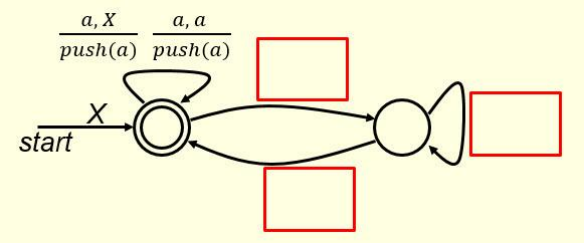
(you only need to answer one case here, either one). The language contains Λ as an element. Now consider the case k = 1 and consider the input string ab. When the first symbol is scanned, we get an 'a'. This information alone is not enough for us to make a proper choice. So we don't even know what to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

- Your solutions must be typed (word processed) and submitted both as a pdf file and a word document to Canvas before 23:59 on 11/13/2026.
- Don't forget to name your files as
  CS375_2026f_HW7_LastName.docx / CS375_2026f_HW7_LastName.pdf