CSE340 Fall 2021 - Homework 1

Due: Wednesday **September 15,** 2021 by 11:59 PM on Gradescope

Note the due date which is different from the original planned due date for HW1

All submissions should be typed. Exception can only be made for drawing parse trees, which can be hand drawn and scanned in the submitted document.

When you submit your solution on Gradescope, you should indicate for each problem the page on which the solution is.

**Remember that no late submissions are accepted for homework assignments.**

**Problem 1.** Consider the list of tokens

T1 = { “supper”, “abcd1e” }

T2 = { “abd” }

SUPPERID = strings that are ID (see below) and that have “supper” as prefix. For example, “supper123”, “supper”, “supper\_”

ID = Set of strings that consist of a letter or underscore that is followed by zero or more letters or digits. For example “\_”, “\_11” are ID but “\_\_” is not an ID according to this definition

NUM = Set of strings that consist of a non-zero digit that is followed by 1 or more digits or the string “0”.

When identifying tokens in the input, we treat & as a separator. This means that getToken() should stop when it reaches & and returns the longest matching prefix up to but not including the &. & itself cannot be part of a token. & is the only separator. Space characters are not separators for this problem.

Consider the input

supper22abcd1\_11&supper\_11&123abcd1e

SUPPERID ID T1 ID NUM T1

t1 t2 t3 t4 t5 t6

t7 t8

and the following sequence of calls:

t1 = lexer.GetToken();

t2 = lexer.GetToken();

t3 = lexer.peek(1);

t4 = lexer.peek(3);

t5 = lexer.peek(5);

t6 = lexer.peek(7);

t7 = lexer.GetToken();

t8 = lexer.GetToken();

**What are the values of t1, t2, t3, t4, t5, t6, t7 and t8?**

**t1**: {SUPPERID, “supper22abcd1”}

There are three possible tokens: T1 whose lexeme is “supper”

SUPPERID whose lexeme is “supper22abcd1”

ID whose lexeme is “supper22abcd1”

The longest possible match is “supper22abcd1”. Since, SUPPERID appears before ID in the list, we return SUPPERID.

**t2**: {ID, “\_11”}

The only possible match is ID whose lexeme is “\_11”. Note that we cannot include the & or skip the & because it is a separator.

**t3**: {T1, “supper”}

Here there are three possible tokens T1, SUPPERID, and ID. Since T1 is listed first, T1 is returned.

**t4**: {NUM, "123"}

Here we are peeking 3 ahead. The only tokens that are consumed are the SUPPERID and the ID at the beginning (**t1** and **t2**). Since we are peeking 3 ahead, we skip the two tokens after t2 and we return the 3rd one.

**t5**: {EOF, “”}

Here we are peeking 5 ahead which takes us beyond the end of the list of tokens, so EOF is returned.

**t6**: {EOF, “”}

Here we are peeking 7 ahead which takes us beyond the end of the list of tokens, so EOF is returned.

**t7**: {T1, “supper”}

Here we resume lexer.GetToken() and we get the next token that is not consumed,

**t8**: {ID, "\_11"}

We return the token aftwr the last token that was consumed.

**Problem 2.** Consider the grammar

S → C D C | D E A

A → a A | ε

C → a C b| ε

D → a D| a

E → b E | a

Assume that the conventions we discussed in class for writing grammars are followed (see notes number 2).

1. What is the start symbol? Explain briefly. No explanation, no credit!

The start symbol is S. By convention, and unless otherwise specified, the left-hand side of the first rule is the start symbol

1. What are the non-terminals? Explain briefly. No explanation, no credit!

The non-terminals are S, A, C, D and E. By convention, and unless otherwise specified, the left-hand side of the rules are the non-terminals.

1. What are the terminals? Explain briefly. No explanation, no credit!

The terminals are a and b. By convention, and unless otherwise specified, the symbols, other than epsilon, that are not non-terminals are terminals. Epsilon is not a terminal. It does not correspond to a token

1. Give two different left-most derivations for the input:

a a a a a a b b b b b b a a

The first left-most derivation should start with S ⇒ C D C

S ⇒ C D C

⇒ a C b D C

⇒ a a C b b D C

⇒ a a a C b b b D C

⇒ a a a a C b b b b D C

⇒ a a a a a C b b b b b D C

⇒ a a a a a a C b b b b b b D C

⇒ a a a a a a Ɛ b b b b b b D C

⇒ a a a a a a b b b b b b a D C

⇒ a a a a a a b b b b b b a a C

⇒ a a a a a a b b b b b b a a Ɛ

= a a a a a a b b b b b b a a

The second left-most derivation should start with S ⇒ D E A

S ⇒ D E A

⇒ a D E A

⇒ a a D E A

⇒ a a a D E A

⇒ a a a a D E A

⇒ a a a a a D E A

⇒ a a a a a a E A

⇒ a a a a a a b E A

⇒ a a a a a a b b E A

⇒ a a a a a a b b b E A

⇒ a a a a a a b b b b E A

⇒ a a a a a a b b b b b E A

⇒ a a a a a a b b b b b b E A

⇒ a a a a a a b b b b b b a A

⇒ a a a a a a b b b b b b a a A

⇒ a a a a a a b b b b b b a a Ɛ

= a a a a a a b b b b b b a a

1. In the left-most derivation of

a a a a a a b b b b b b a a

that starts with S => C D C, which part of the input corresponds to the first C? which part of the input corresponds to the D and which part of the input corresponds to the second C? You answer should have the following format:

The part of input that corresponds to the first C is …

aaaaaabbbbbb

The part of input that corresponds to the D is …

aa

The part of input that corresponds to the second C is …

ε

a a a a a a b b b b b b a a

C D C

1. What is the language of this grammar (remember that the language of a grammar is the set of strings that can be derived from the start symbol or, equivalently, the set of strings that have parse trees according to the grammar). In your description of the language, you should try to be as precise but as brief as possible. This part is a little tricky!

Set of strings that can be generated from A = { a*i* : *i* ≥ 0 }

Set of strings that can be generated from C = { a*i* b*i* : *i* ≥ 0 }

Set of strings that can be generated from D = { a*i* : *i* > 0 }

Set of strings that can be generated from E = { b*i* a : *i* ≥ 0 }

L(G) = { a*i* b*i* a*j* a*k* b*k* : *i*,*k* ≥ 0 , *j > 0* } ∪ { a*i* b*j* a a*k* : *i* > 0,  *j,k*  ≥ 0 }

**Problem 3.** Consider the grammar

A → Y Y Z | Z Y Y

X → b B | Z

B → a B | ε

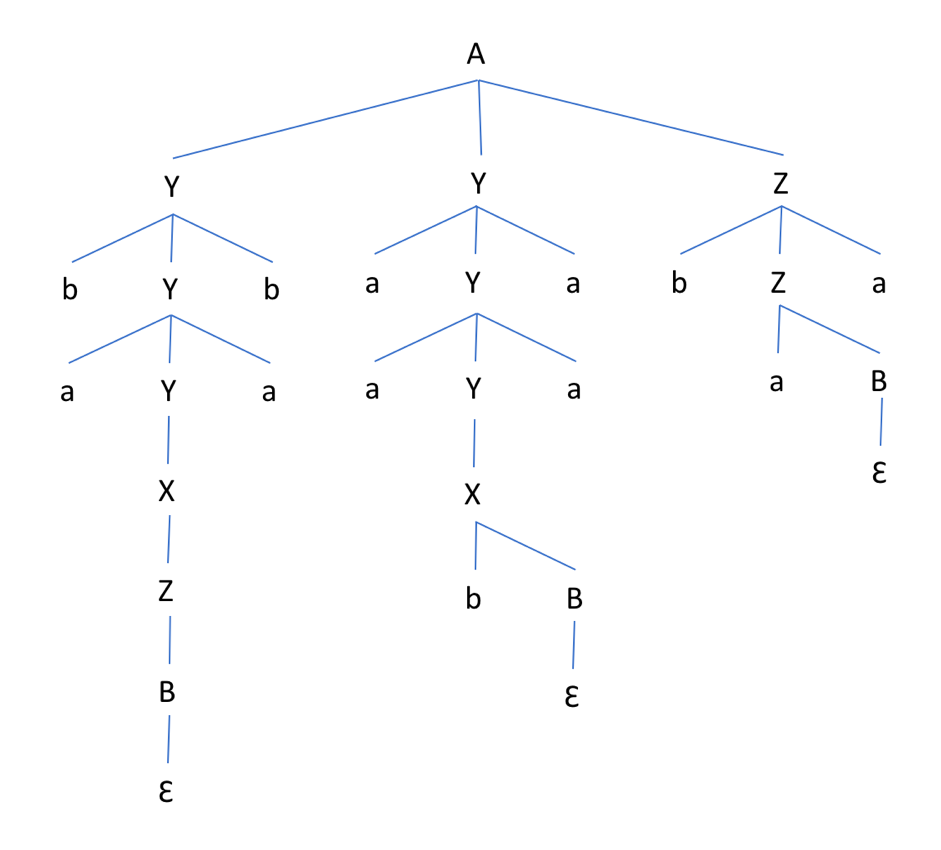
Y → a Y a | b Y b | X

Z → b Z a | B

where A, B, X, Y and Z are non-terminal, A is the start symbol and a and b are tokens. Remember that ε represent the empty sequence. Y → ε means that Y does not have to match any tokens or, equivalently, it matches an empty sequence of tokens.

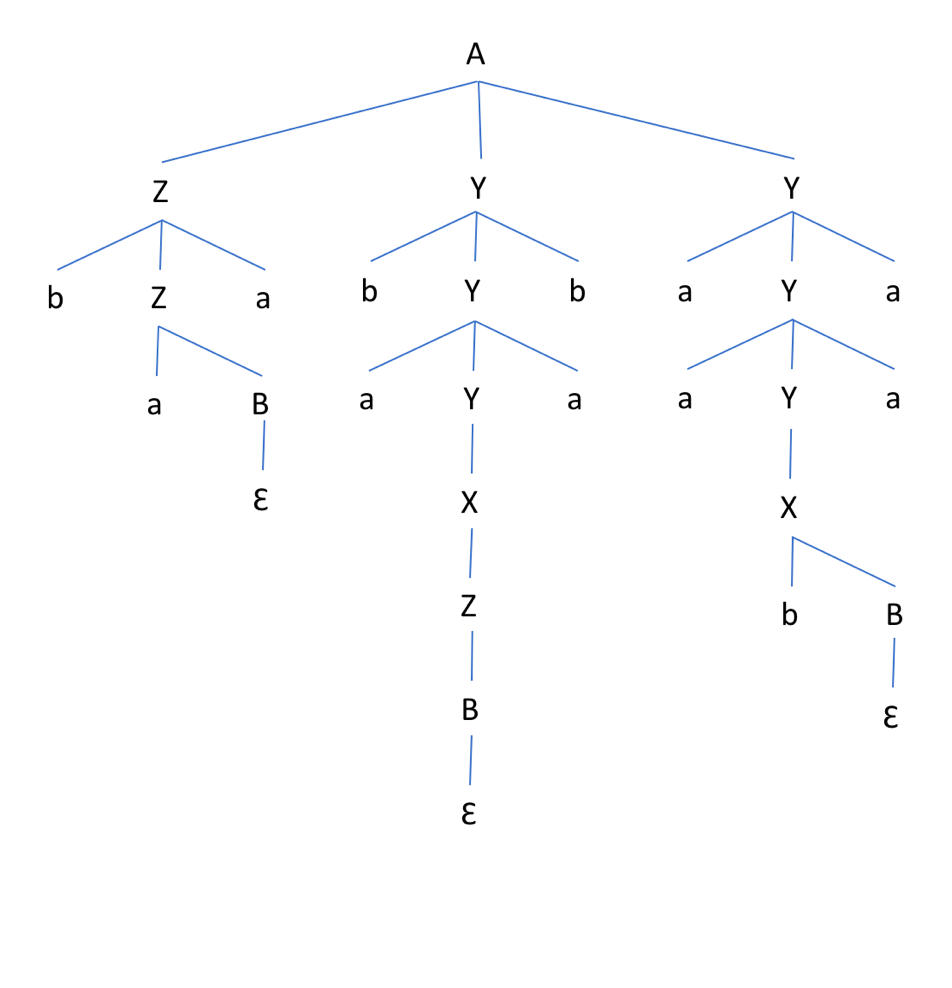
1. Give a parse tree for the sequence of tokens:

**b a a b a a b a a b a a**

****

1. Give a another (different) parse tree for the sequence of tokens:

**b a a b a a b a a b a a**



1. Is this grammar ambiguous? Why?

This grammar is ambiguous because the input b a a b a a b a a b a a has two different parse trees.