Note: in all of the below capital letters are non-terminals, non-grammar symbols are terminals.

0. Given the following grammar, G:

  S → A | S # S | A @ A
  A → C | C A
  C → a | b | c

  For each production below, state whether it is:

    In the language G decides

      - if so, prove it with a derviation and parse tree
      - if not, explain why

    If it is ambiguous in G

      - if so, prove it is with another valid parse tree in G

  Productions:

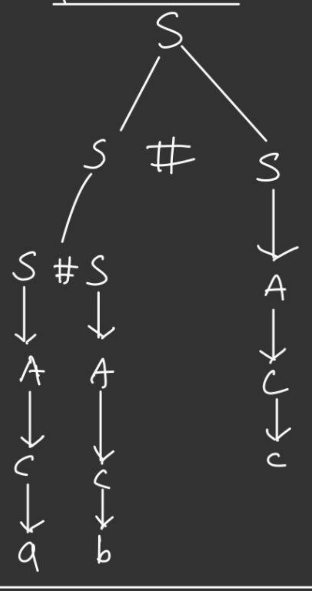
    I. a # b # c
    II. a @ b # c
    III. a @ b @ c

<u>Answers Are On The NEXT Page!!!</u>

CS314 – Principles of Programming Languages — January 31st, 2022

0.) I. a # b # c

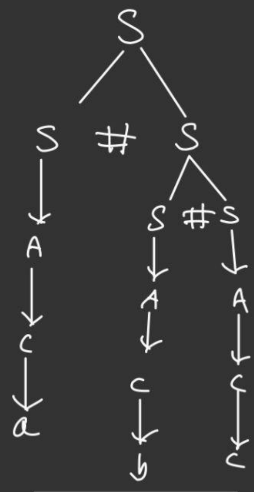
Parse Tree



Derivation

$S \to S\#S$
$= S\#S\#S$
$= A\#A\#A$
$= C\#C\#C$
$= a\#b\#c$

Parse Tree #2

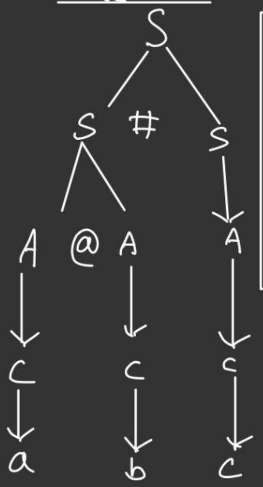


AMBIGOUS

This production is ambigous. Based on the second parse tree ambiguity is present in the given grammar.

II. a @ b # c

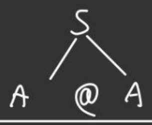
Parse Tree



Derivation

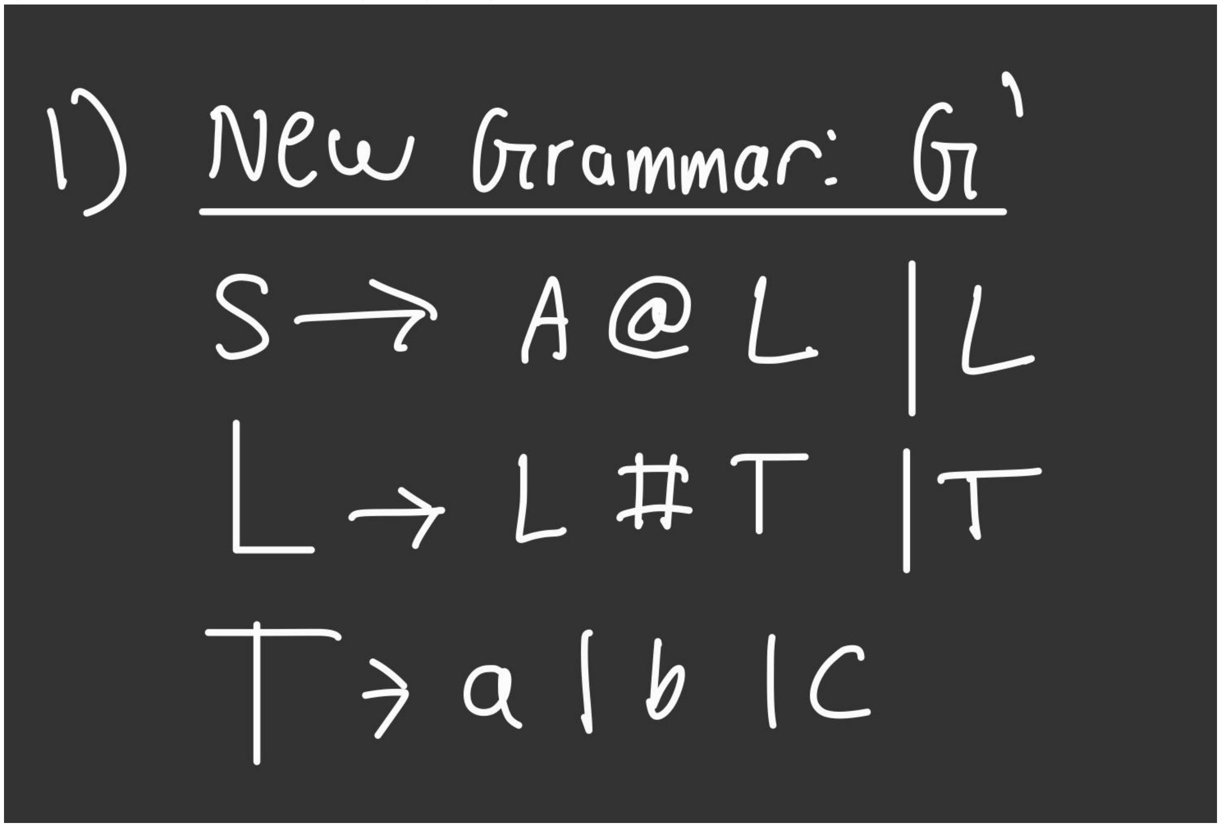
$S \to S\#S$
$= A@A\#S$
$= A@A\#A$
$= C@A\#A$
$= C@C\#C$
$= a@b\#C$

III. a @ b @ c



cannot be derived since there is no way to get 3 non-terminals in the format desired since A has no terminal symbols. Thus, there is no way to derive or parse this string.

1. Rewrite grammar G so that:
   - no string in it is ambiguous
   - # has higher precedence than @
   - # and @ are both left-associative
   (i.e. a # b # c should mean (a # b) # c)



1) New Grammar: G'

$S \rightarrow A @ L \mid L$

$L \rightarrow L \# T \mid T$

$T \rightarrow a \mid b \mid c$

(I was not sure about the variable naming conventions when using grammar, so I just used obvious capital letters.)

2. Given grammar H below:

(in Extended BNF, terminals in bold and in red):

<Statement> → <Assignment>

<Statement> → <While>

<Assignment> → <Var> **=** <Value> [**,** <Value>] **;**

<While> → **while(**<Value>**)** **{** {<Statement>} **}**

<Value> → <Var> | <Number>

<Var> → **a** | **b** | **c**

<Number> → **0** | **1**

For each production below:

- determine if it is in the language H
- explain why or why not

Productions:

I. 0 = a,b;

This is not in the language since it is defined that a variable must be either *a* or *b* or *c*. Since the variable is 0 in this language, it can be concluded that this statement is not valid in this grammar.

II. a = 0,b;

This is valid because the variable in this case, *a*, is a valid variable in this grammar. Then, based on the assignment definition of the grammar, it must be a <var> followed by an equal sign, then a value defined by a number, followed by an optional symbol and non-terminal. The optional symbol is a comma, which is present, and that is followed by a non-terminal which is also present as a <value>. A value can be either a variable or number. Since it is *b*, this is a valid variable making it a valid statement in the grammar.

III. while(a = 1,0;){b = 0;}
This grammar is not valid because in the parentheses of the <while> non-terminal, we encountered an unexpected value. We were only expecting one value, but in this statement, we see there are two values given within the parentheses of the while loop which is invalid. This grammar is not structured correctly to handle two values within the <while> non-terminal. In this grammar, only assignments can have two values, <while> cannot.
IV. while(a) {while (a = 0;)}
This statement is also not valid in the given grammar because in the second <while>, we see a value being assigned. This is invalid in our grammar. We cannot have an assignment within the parentheses of the <while> non-terminal. This is a violation of our grammar and thus is not valid.