Chapter 3: Describing Syntax and Semantics

1. What is the difference between a sentence and a sentential form?
Ans:

- A sentence is a sentential form that has only terminal symbols. On the other hand, every string of symbols in a derivation is a sentential form.
- Every sentence is a sentential form but every sentential form is not a sentence.

2. Describe the basic concept of operational Semantics.
Ans:

- Operational Semantics describe the meaning of a program by executing its statements on a machine, either simulated or actual.
- The change in the state of the machine (memory, registers, etc.) defines the meaning of the statement.
- To use operational semantics for a high-level language, a virtual machine is needed.
- Uses: (i) Language manuals and textbooks (ii) Teaching programming languages.

3. Define syntax and semantics.
Ans:
Syntax: Syntax is the form or structure of the expressions, statements, and program units.
Semantics: Semantics is the meaning of the expressions, statements, and program units.
Syntax and semantics provide a language's definition

4. What is the difference between a synthesized and an inherited attribute?
Ans:

Synthesized Attribute	Inherited Attribute
An attribute is said to be Synthesized attribute if its parse tree node value is determined by the attribute value at child nodes.	An attribute is said to be Inherited attribute if its parse tree node value is determined by the attribute value at parent and/or siblings' node.
If all attributes are synthesized, the tree can be decorated in bottom-up order.	If all attributes are inherited, the tree can be decorated in top-down order.
The production must have non-terminal as its head.	The production must have non-terminal as a symbol in its body.

5. Is aabcdccc in the language generated by the following grammar? If so, draw the parse tree. If not, show why.

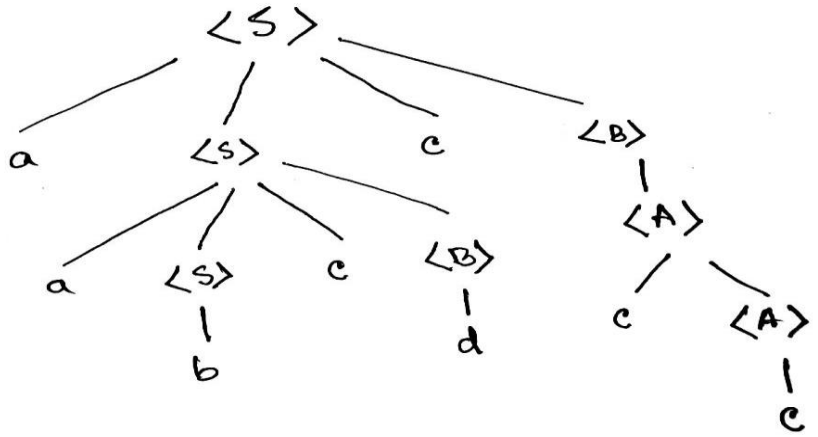
<S> → a <S> c <B> | <A> | b

<A> → c <A> | c

<B> → d | <A>

Ans: Yes, aabcdccc in the language is generated by the following grammar.

Parse tree:



6. Is aacccdcc in the language generated by the following grammar? If so, draw the parse tree. If not, show why.

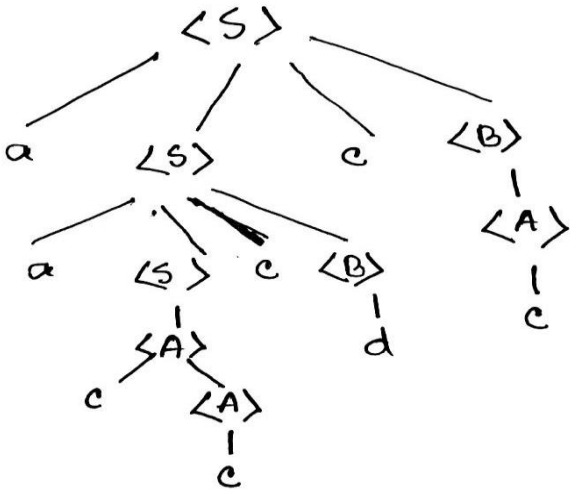
<S> → a <S> c <B> | <A> | b

<A> → c <A> | c

<B> → d | <A>

Ans: Yes, aacccdcc in the language is generated by the following grammar.

Parse tree:



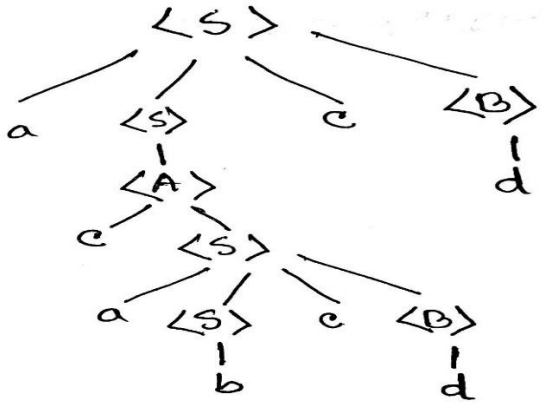
7. Derive the string 'acabcdcd' using the following grammar (make leftmost derivation):

<S> → a <S> c <B> | <A> | b
<A> → c <S> | c
<B> → d | b<A>

Ans: Derivation of the string 'acabcdcd':

<S> → a<S>c<B>
→ a<A>c<B> [Using <S> → <A>]
→ ac<S>c<B> [Using <A> → c<S>]
→ aca<S>c<B>c<B> [Using <S> → a<S>c<B>]
→ acabc<B>c<B> [Using <S> → b]
→ acabcdc<B> [Using <B> → d]
→ acabcdcd [Using <B> → d]
(derived)

using the Parse tree:



8. Prove that the following grammar is ambiguous:

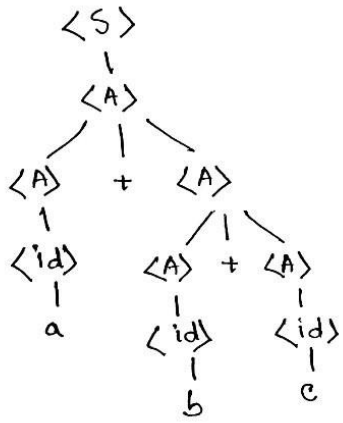
<S> → <A>

<A> → <A> + <A> | <id>

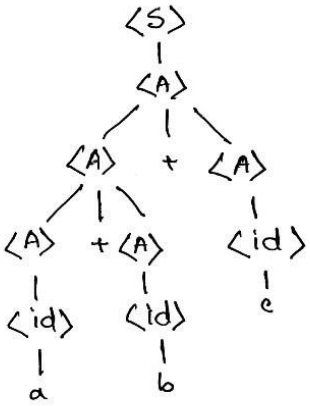
<id> → a | b | c

Ans: If a grammar generates different parse tree, then we can say that, the grammar is ambiguous means it is not a good grammar. The given grammar generates two different parse trees. The parse trees are shown below.

Parse Tree-I



Parse Tree-II



So, the given grammar is ambiguous.

9. Consider the following grammar:

<S> → a <S> c <B> | <A> | b

<A> → c <A> | c

<B> → d | <A>

Which of the following sentences are in the language generated by this grammar?

a. abcd

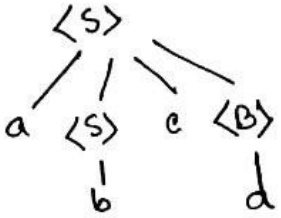
b. acccbd

c. acccbcc

d. acd

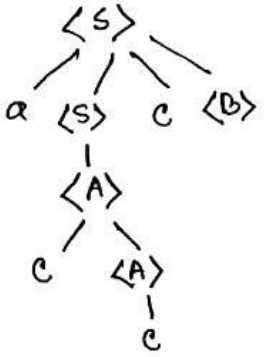
e. accc

(a) abcd :



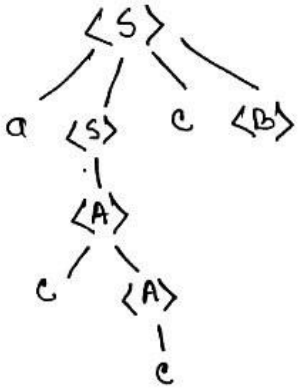
so, the sentence 'abcd' is in the language generated by this grammar.

(b) acccbd:



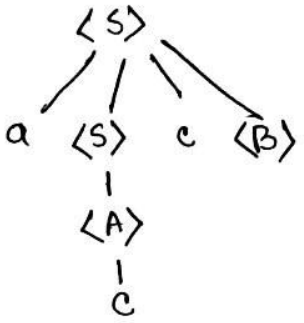
so, the sentence 'acccbd' isn't in the language generated by this grammar. the parse tree can't generate the sentence.

(c) acccbcc:



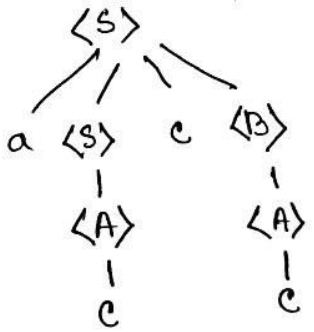
so, the sentence 'acccbcc' isn't in the language generated by this grammar. the parse tree can't generate the sentence.

(d) acd:



So, the sentence 'acd' isn't in the language generated by this grammar. The parse tree can't generate the sentence.

(e) accc:



So, the sentence 'accc' is in the language generated by this grammar.

10. Convert BNF to EBNF

(i) BNF: <expr> → <expr> + <term>
| <expr> - <term>
| <term>

EBNF: <expr> → <term> { (+ | -) <term> }

(ii) BNF: <term> → <term> * <factor>
| <term> / <factor>
| <factor>

EBNF: <term> → <factor> { (* | /) <factor> }

(iii) BNF: ~~begin~~ <program> → begin <stmt_list> end

EBNF: <program> → begin <stmt_list> end

(iv) BNF: <stmt_list> → <stmt>
| <stmt> ; <stmt_list>

EBNF: <stmt_list> → <stmt> { ; <stmt_list> }

(v) ~~Stmt~~ BNF: <Stmt> → <var> = <expression>

EBNF: <Stmt> → <Var> = <expression>

(vi) BNF: <var> → A | B | C

EBNF: <var> → A | B | C

(vii) BNF: <expression> → <var> + <var>
| <var> - <var>
| <var>

EBNF: <expression> → <var> { (+ | -) <var> }

(VIII) BNF: <assign> → <id> = <expr>

EBNF: <assign> → <id> = <expr>

(IX) BNF: <id> → A | B | C

EBNF: <id> → A | B | C

(X) ~~<expr~~ BNF: <expr> → <expr> + <expr>
| <expr> * <expr>
| <expr>
| <id>

EBNF: <expr> → <expr> { (+ | *) <expr> }
| <id>