NORTH SOUTH UNIVERSITY

Department of Electrical and Computer Engineering

Homework – 02

Name	: Joy Kumar Ghosh
Student ID	: 2211424 6 42
Course No.	: CSE 425
Course Title	: Concepts of Programming Language
Section	: 1
Date	: 31 October 2024

Ans. to the ques.no.01

Lexemes:

Lexemes is a string of characters that are the lowest syntactical unit. Includes numerical literals, identifiers, reserve words.

Token:

Token is a group or collection of lexemes given a name. Sometimes, a single lexemes can also form a token.

Example:

Source Code:

```
while (x ≥ 5):
    x = x - 6;
```

Lexemes	Tokens
while	KEYWORDS
(	LPAREN
x	ID
≥	OP
5	NUMBER
)	RPAREN
:	CLONE
x	ID
=	EQ
x	ID
-	OP
6	NUMBER
;	SEMICLONE

Reserved Words:

Some words are keep reserved as an identifier for various operations, these words can't be used as user defined identifier.

Example: do, while, if, etc.

do if, individually they are reserved word, but if we write together "doif" then it will be a valid identifier.

Metalanguage:

Metalanguage is a symbolic language that is used to define other language.

| ⇒ OR

→ ⇒ defined as

:: ⇒ defined as

etc.

Context Free Grammar (CFG):

CFG is a formal grammar that describes the syntax of a PL. This grammar defined how tokens can be combined to form valid statements in the PL.

Example of CFG using Metalanguage:

$$\langle number \rangle \longrightarrow \langle number \rangle \langle digit \rangle \mid \langle digit \rangle$$

$$\langle digit \rangle \longrightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Derivation:

Derivation is a process where we replace non-terminal with terminals word one by one. There are two types of derivation:

(i) Left most derivation : left non-terminal replaced first

(ii) Right most derivation : Right non-terminal replaced first

Production Rule:

Some rules defined in CFG that specify how symbols can be replaced by other symbols to form a valid statement in PL.

Example of Derivation:

Grammar:

⟨number⟩ → ⟨number⟩ ⟨digit⟩ | ⟨digit⟩

⟨digit⟩ → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Source / String: 2 3 4

Metalanguage derivation, left most:

⟨number⟩ → ⟨number⟩ ⟨digit⟩

→ ⟨number⟩ ⟨digit⟩ ⟨digit⟩

→ ⟨digit⟩ ⟨digit⟩ ⟨digit⟩

→ 2 ⟨digit⟩ ⟨digit⟩

→ 2 3 ⟨digit⟩

→ 2 3 4

Ans. to the ques. no. 02

Backus-Naur Form (BNF) is a context free grammar form, used to define grammar of a PL. And EBNF is an extended version of BNF, proposed for convenience and reduce the lines on grammar.

In EBNF:

$\{\ldots\}$ ⇒ 0 or any number of repeatation

$(\ldots)$ ⇒ options.

Example:

⟨number⟩ → ⟨number⟩ ⟨digit⟩

→ ⟨number⟩ ⟨digit⟩ ⟨digit⟩

→ ⟨digit⟩ ⟨digit⟩ ⟨digit⟩ ....

EBNF

⟨number⟩ → ⟨digit⟩ { ⟨digit⟩ }

Combined Example:

BNF:

⟨expn⟩ → ⟨expn⟩ + ⟨term⟩ |
⟨expn⟩ - ⟨term⟩ |
⟨term⟩ $

⟨term⟩ → ⟨term⟩ * ⟨factor⟩ |
⟨term⟩ / ⟨factor⟩ |
⟨factor⟩

⟨factor⟩ → ⟨exp⟩ ** ⟨factor⟩ |
⟨exp⟩

⟨exp⟩ → (⟨expn⟩) | id

EBNF:

⟨expn⟩ → ⟨term⟩ { (+|-) ⟨term⟩ }

⟨term⟩ → ⟨factor⟩ { (*|/) ⟨factor⟩ }

⟨factor⟩ → ⟨exp⟩ { ** ⟨factor⟩ }

⟨exp⟩ → (⟨expn⟩) | id

Ans. to the ques. no.03

Given gramar:

$\langle S \rangle \longrightarrow \langle A \rangle\ a\ \langle B \rangle\ b$

$\langle A \rangle \longrightarrow \langle A \rangle b \mid b$

$\langle B \rangle \longrightarrow b$

For string : babb

$S \longrightarrow \langle A \rangle\ a\ \langle B \rangle\ b$

$\longrightarrow b\ a\ \langle B \rangle\ b \quad [\langle A \rangle = b]$

$\longrightarrow b\ a\ b\ b \quad [\langle B \rangle = b]$

$\therefore$ babb is derivable.

For string : bbb a bb

$S \longrightarrow \langle A \rangle\ a\ \langle B \rangle\ b$

$\longrightarrow \langle A \rangle b\ a\ \langle B \rangle\ b \quad [\langle A \rangle = \langle A \rangle b]$

$\longrightarrow \langle A \rangle\ b\ b\ a\ \langle B \rangle b \quad [\langle A \rangle = \langle A \rangle b]$

$\longrightarrow b\ b\ b\ a\ \langle B \rangle\ b \quad [\langle A \rangle = b]$

$\longrightarrow b\ b\ b\ a\ b\ b \quad [\langle B \rangle = b]$

$\therefore$ bbbabb is derivable.

For string: bbaaaaabc

As we can see that there is a terminal "c" which is not defined in the grammar.

Therefore, the string is not derivable.

For string: aaaaaa

"a" is not available as a single terminal in the grammar. As a result, whatever we choose, it will produce b. Therefore, the string aaaaaa is not derivable by the this grammar.

Ans. to the ques. no. 04

The grammar will be,

$\langle S \rangle \rightarrow a\langle S \rangle b$

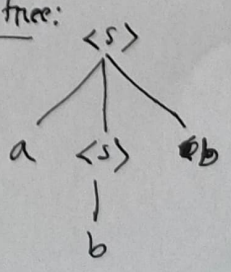
$\langle S \rangle \rightarrow b$

For string: abb

$\langle S \rangle \rightarrow a\langle S \rangle b$

$\rightarrow$ abb $[\langle S \rangle = b]$

Parse tree:



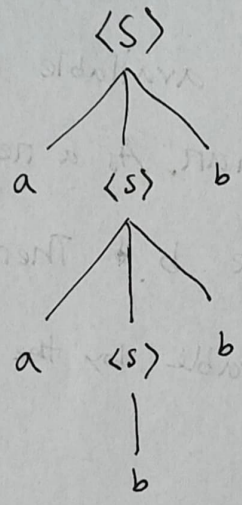
For string: aa bbb

$\langle S \rangle \rightarrow a \langle S \rangle b$

$\rightarrow a\ a \langle S \rangle b\ b \quad [\langle S \rangle = a \langle S \rangle b]$

$\rightarrow a\ a\ b\ b\ b \quad [\langle S \rangle = b]$

Parse tree:



Ans. to the ques. no. 05

Definition of Identifer:

⟨identifier⟩ → ⟨letter⟩ ⟨rest⟩

⟨rest⟩ → ⟨letter⟩ ⟨rest⟩ | ⟨digit⟩⟨rest⟩ | ∈

⟨letter⟩ → a|b|...|y|z|A|B|....|Y|Z

⟨digit⟩ → 0|1|2|3|...|8|9

Ans. to the ques. no. 06

If a designed grammar has more than one left-most derivation or right-most derivation, then the grammar is ambiguous.

Example:

Grammar:

$$S \longrightarrow AS \mid \epsilon$$

$$A \longrightarrow A1 \mid 0A1 \mid 01$$

String: 00111

Left-most derivation:

$S \longrightarrow AS$

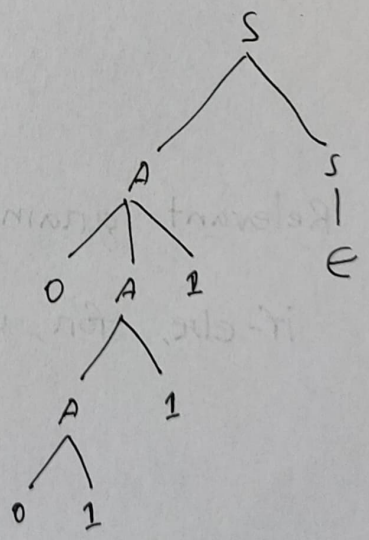
$\longrightarrow 0A1S$ [A = 0A1]

$\longrightarrow 0A11S$ [A = A1]

$\longrightarrow 00111S$ [A = 01]

$\longrightarrow 00111$ [S = ϵ]

Parse tree:



Another left-most derivation:

$S \rightarrow AS$

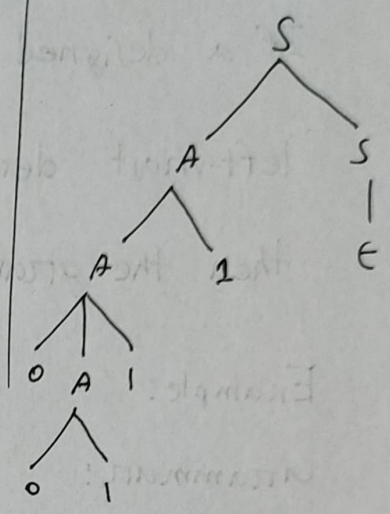
$\rightarrow AIS \quad [A = AI]$

$\rightarrow 0AIIS \quad [A = 0AI]$

$\rightarrow 001IIS \quad [A = 0I]$

$\rightarrow 00111 \quad [S = \epsilon]$

Parse tree:



This grammar has two left-most derivation for the string "00111".

Therefore, the grammar is ~~any~~ ambiguous.

Ans. to the ques. no. 07

Relevant grammars of three control statements if-else, for, while are given bellow:

Grammar for if-else statement:

⟨IfStmt⟩ → if (⟨condition⟩) ⟨statement⟩ ⟨else⟩

⟨else⟩ → else ~~if~~ ⟨statement⟩ | ∈ | else ⟨IfStmt⟩

⟨condition⟩ → ⟨expr⟩

⟨statement⟩ → ⟨expr⟩ ; | {⟨block⟩}

⟨block⟩ → ⟨IfStmt⟩ | ⟨ForStmt⟩ | ⟨WhileStmt⟩ | ⟨stmtlist⟩

⟨stmtlist⟩ → ⟨statement⟩ ⟨stmtlist⟩ | ∈

Grammar for "for" statement:

⟨ForStmt⟩ → for (⟨init⟩ ; ⟨condition⟩ ; ⟨update⟩) ⟨statement⟩

⟨init⟩ → ⟨assignment⟩ | ⟨expr⟩ | ∈

⟨condition⟩ → ⟨expr⟩ | ∈

⟨update⟩ → ⟨expr⟩ | ∈

⟨statement⟩ → ⟨expr⟩ ; | { ⟨block⟩ }

⟨block⟩ → ⟨IfStmt⟩ | ⟨ForStmt⟩ | ⟨WhileStmt⟩ | ⟨stmtlist⟩

⟨stmtlist⟩ → ⟨statement⟩ ⟨stmtlist⟩ | ∈

Grammar for while statement:

⟨WhileStmt⟩ → while (⟨condition⟩) ⟨statement⟩

⟨condition⟩ → ⟨expr⟩

⟨statement⟩ → ⟨expr⟩ ; | { ⟨block⟩ }

⟨block⟩ → ⟨IfStmt⟩ | ⟨ForStmt⟩ | ⟨WhileStmt⟩ | ⟨StmtList⟩

⟨StmtList⟩ → ⟨statement⟩ ⟨StmtList⟩ | ∈

Ref.:

- Class Notes
- Handouts provided by ~~MSK1~~ Dr. Md Shahriar Karim [MSK1]
- ChatGPT - for the last question.