Experiment No.2

Title: - Left Recursion

Aim:- Write a program to eliminate direct and indirect left recursion.

Left Recursion :-

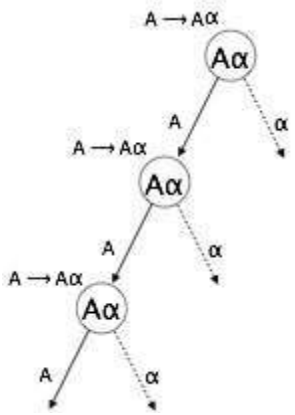
A grammar becomes left-recursive if it has any non-terminal ‘A’ whose derivation contains ‘A’ itself as the left-most symbol. Left-recursive grammar is considered to be a problematic situation for top-down parsers. Top-down parsers start parsing from the Start symbol, which in itself is non-terminal. So, when the parser encounters the same non-terminal in its derivation, it becomes hard for it to judge when to stop parsing the left non-terminal and it goes into an infinite loop.

Example:

```
(1) A => Aα | β

(2) S => Aα | β
    A => Sd
```

(1) is an example of immediate left recursion, where A is any non-terminal symbol and α represents a string of non-terminals.
(2) is an example of indirect-left recursion.



A top-down parser will first parse the A, which in-turn will yield a string consisting of A itself and the parser may go into a loop forever.

Removal of Left Recursion

One way to remove left recursion is to use the following technique:
The production

```
A => Aα | β
```

is converted into following productions

```
A => βA'
A'=> αA' | ε
```

This does not impact the strings derived from the grammar, but it removes immediate left recursion.
Second method is to use the following algorithm, which should eliminate all direct and indirect left recursions.

```
START

Arrange non-terminals in some order like A1, A2, A3,…, An

   for each i from 1 to n
      {
      for each j from 1 to i-1
         {
         replace each production of form Ai ⟹Aj𝛾
         with Ai ⟹ δ1𝛾  | δ2𝛾 | δ3𝛾 |…| 𝛾
         where Aj ⟹ δ1 | δ2|…| δn  are current Aj productions
         }
      }
   eliminate immediate left-recursion

END
```

Example

The production set

```
S => Aα | β
A => Sd
```

after applying the above algorithm, should become

```
S => Aα | β
A => Aαd | βd
```

and then, remove immediate left recursion using the first technique.

```
A  => βdA'
A' => αdA' | ε
```

Now none of the production has either direct or indirect left recursion.

Exercise:-

Q1. The following grammar can be used to describe traveling schemes:

TS → TS Time Time TS | Station

Station → Identifier

Time → Nat : Nat

Which of the following grammars is equivalent but no longer left-recursive? **(Tick Mark right answer)**

a) TS → TS (Time Time Station) ∗

Station → Identifier

Time → Nat : Nat

b) TS → Station | Station Z

Z → Time Time TS | Time Time TS Z

Station → Identifier

Time → Nat : Nat

c) TS → Z Time Time TS | Station

Z → TS | #

Station → Identifier

Time → Nat : Nat

d) TS → Station Time Time Z

Z → Station | TS

Station → Identifier

Time → Nat : Nat

Q2. Consider the grammar

S → SX | SSb | XS | a
X → Xb | Sa | b
Eliminate left recursion and rewrite the grammar.

Answer:-

Q3. Is the resulting grammar of Q2. suitable for top down parsing?

Answer: