**Removing Direct and Indirect Left Recursion in a Grammar**

* Difficulty Level : [Hard](https://www.geeksforgeeks.org/hard/)
* Last Updated : 05 Dec, 2022

Prerequisite – [Classification of Context Free Grammars](https://www.geeksforgeeks.org/classification-of-context-free-grammars/), [Ambiguity and Parsers](https://www.geeksforgeeks.org/parsing-set-1-introduction-ambiguity-and-parsers/)

**Left Recursion:** Grammar of the form,

*S* ⇒ *S* | a | b

is called *left recursive* where *S i*s any non Terminal and a and b are any set of terminals.

**Problem with Left Recursion:** If a left recursion is present in any grammar then, during parsing in the syntax analysis part of compilation, there is a chance that the grammar will create an infinite loop. This is because, at every time of production of grammar, S will produce another S without checking any condition.

**Algorithm to Remove Left Recursion with an example:** Suppose we have a grammar which contains left recursion:

S ⇒ S a | S b | c | d

Check if the given grammar contains left recursion. If present, then separate the production and start working on it.  In our example:

S ⇒ S a | S b | c | d

Introduce a new nonterminal and write it at the end of every terminal. We create a new nonterminal **S’**and write the new production as:

S ⇒ c**S'** | d**S'**

Write the newly produced nonterminal **S’** in the LHS, and in the RHS it can either produce **S’** or it can produce new production in which the terminals or non terminals which followed the previous LHS will be replaced by the new nonterminal **S’** at the end of the term.

**S'** ⇒ ε | a**S'** | b**S'**

So, after conversion, the new equivalent production is:

S ⇒ c**S'** | d**S'**

**S'** ⇒ ε | a**S'** | b**S'**

**Indirect Left Recursion:** A grammar is said to have indirect left recursion if, starting from any symbol of the grammar, it is possible to derive a string whose head is that symbol. For example,

***A*** ⇒ B r

B ⇒ C d

C ⇒ ***A*** t

where A, B, C are non-terminals and r, d, t are terminals. Here, starting with A, we can derive A again by substituting C to B and B to A.

**Algorithm to remove Indirect Recursion with help of an example:**

A1 ⇒ A2 A3

A2 ⇒ A3 A1 | b

A3 ⇒ A1 A1 | a

Where A1, A2, A3 are non terminals and a, b are terminals.

Identify the productions which can cause indirect left recursion. In our case,

A3 ⇒ A1 A1 | a

Substitute its production at the place the terminal is present in any other production: substitute A1–> A2 A3 in production of A3.

A3 ⇒ A2 A3 A1 | a

Now in this production substitute A2 ⇒ A3 A1 | b

A3 ⇒ (A3 A1 | b) A3 A1 | a

and then distributing,

A3 ⇒ A3 A1 A3 A1 | b A3 A1 | a

Now the new production is converted in the form of direct left recursion, solve this by the direct left recursion method.

Eliminating direct left recursion as in the above, introduce a new nonterminal and write it at the end of every terminal. We create a new nonterminal **A’** and write the new productions as:

A3 ⇒ b A3 A1 ***A'*** | a***A'***

***A'*** ⇒ ε | A1 A3 A1 ***A'***

ε can be distributed to avoid an empty term:

A3 ⇒ b A3 A1 | a | b A3 A1 ***A'*** | a***A'***

***A'*** ⇒ A1 A3 A1 | A1 A3 A1 ***A'***

The resulting grammar is then:

A1 ⇒ A2 A3

A2 ⇒ A3 A1 | b

A3 ⇒ b A3 A1 | a | b A3 A1 ***A'*** | a***A'***

***A'*** ⇒ A1 A3 A1 | A1 A3 A1 ***A'***

Related Articles