**Compiler Construction Experiment 03**

Shubham Golwal | 2020300015 | TE COMPS

23-02-23

**AIM:** Program to remove left recursion for the given grammar.

**THEORY:**

What is lefty recursion?

Left recursion is a grammar production in which a nonterminal symbol A can produce a sequence of symbols that begins with itself, such as A → Aα or A → Aα | β, where α and β are any sequences of symbols, including possibly empty sequences (ε).

When a grammar contains left recursion, it can cause problems for certain parsing algorithms and can make the grammar ambiguous, which can lead to difficulties in understanding and implementing the grammar.

To avoid left recursion, we can apply various techniques to modify the grammar, such as left recursion elimination, which rewrites the production in a way that avoids left recursion.

To remove left recursion from a grammar, we can follow the following general steps:

1. Identify the left-recursive nonterminals in the grammar.
2. For each left-recursive nonterminal A, create a new nonterminal A' and rewrite the productions for A in terms of A' (if any). This means that for each production A → Aα | β, we replace it with A → βA', A' → αA' | ε.
3. Replace any productions that refer to A in the original grammar with productions that refer to A' instead.

For example, let's consider the following grammar with left recursion:

A → Aα | β

We can remove left recursion from this grammar by creating a new nonterminal A' and rewriting the production for A as follows:

A → βA'

A' → αA' | ε

So the resulting grammar without left recursion is:

A → βA'

A' → αA' | ε

Note that this is just a simple example, and the actual process for removing left recursion from a grammar may be more complicated, depending on the structure of the original grammar. However, the general steps above can be applied to any left-recursive grammar to eliminate left recursion.

**CODE:**

import java.util.ArrayList;

//for production give it in aA | Aa' format that is between productions ' | ' should be there with gap

// else will give wrong answer

class left\_recursion{

public static *void* main(String[] *args*) {

ArrayList<String> ar=**new** ArrayList<>();

ar.add(0, "S->Sa | aS | bS | Sbb");

ar.add("A->aA | All");

remove\_direct(ar);

}

public static *void* remove\_direct(ArrayList<String> *production*){

ArrayList<String> fin=**new** ArrayList<>();

for(String s: *production*){

ArrayList<String> beta=**new** ArrayList<>();

ArrayList<String> recursion=**new** ArrayList<>();

String[] array=s.split("->",0); // this splits the first character (LHS) and RHS

String first=array[0]; //Leading charcter

String S=array[0]+"'";// Making prime of that for leading character

String[] right=array[1].split(" | ",0);//splitting the production

for(String re: right){

if(re.equals("|")) continue; //ignore this sign

if(first.charAt(0)==re.charAt(0)){ //this checks is there is left recursion present

recursion.add(re); //adds it to recursion arraylist

}else{

beta.add(re); //adds it to beta as ther's no left recursion

}

}

String betaProd=""; //this willl store all the beta productions in a single string

for(String x: beta){

betaProd+=" | "+x+S;//appending the prime of leading character

}

fin.add(array[0]+" ->"+betaProd.substring(3));// first three characters are: ' | ' hence removing that

String prod="";

for(String x: recursion){

*int* counter=-1;

while(array[0].charAt(0)==x.charAt(++counter));//this counter number of left recursions

*int* counterCopy=counter;

String f="";

while(counter!=0) {

f+=S;

counter--;

}

prod+=x.substring(counterCopy)+f;

prod+=" | ";

}

prod+="epsilon"; //adding epsilon to prime production

fin.add(S+"->"+prod);

System.out.println("Initial production: ");

System.out.println(s);

System.out.println("Final");

for(String x: fin){

System.out.println(x);

}

}

}

static *void* printArrayList(ArrayList<String> *ar*,String *s*){

System.out.println(*s*);

System.out.println("Values");

for(String ss: *ar*){

System.out.println(ss);

}

}

}

Indirect method:

import java.util.\*;

import java.io.\*;

class LR {

static Map<String,String> prod = new HashMap<>();

static Map<String,String> newprod = new HashMap<>();

static Vector<String> alpha = new Vector<>();

static Vector<String> beta = new Vector<>();

public static void main(String[] args) {

int i;

String a,b;

System.out.println("Enter number of productions:");

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

System.out.println("Enter productions: ");

for(i=0;i<n;i++) {

a=sc.next();

b=sc.next();

prod.put(a,b);

}

Iterator it = prod.keySet().iterator();

while(it.hasNext()) {

int pos=0,flag=0;i=0;

String key = (String) it.next();

String value = prod.get(key);

String[] prods = value.split("\\|");

for(String p:prods) {

if(p.charAt(0) == key.charAt(0)) alpha.add(p.substring(1));

else beta.add(p);

pos++;

}

if(alpha.size()>0) {

String valkey="";

for(String al:alpha) {

valkey+=al+key+"'|";

}

valkey+="e";

newprod.put(key+"'",valkey);

valkey="";

for(String be:beta) {

valkey+=be+key+"'|";

}

newprod.put(key,valkey);

}

else {

newprod.put(key,value);

}

}

it = newprod.keySet().iterator();

while(it.hasNext()) {

String key = (String) it.next();

System.out.println(key+" "+newprod.get(key));

}

}

}

**Input:**

**Output:**

**CONCLUSION:**

**REF.:**