

03-Mar-22

Experiment 6 - Predictive Parsing Table

Dhawal Patil

RA1911003010575

CSE A2

Experiment 6

Aim:

To write a program to show predictive parsing table using Python language.

Algorithm:

1. Start the program
2. Initialize the required variables
3. Get the number of coordinates and productions from the user
4. Perform the following:
 - For (each production $A \rightarrow \alpha$ in G)
 - For (each terminal a in $FIRST(\alpha)$)
 - Add $A \rightarrow \alpha$ to $M[A,a]$;
 - If (ϵ is in $FIRST(\alpha)$)
 - For (each symbol b in $FOLLOW(A)$)
 - Add $A \rightarrow \alpha$ to $M[A,b]$
5. Print the resulting stack
6. Print if the grammar is accepted or not
7. Exit the program

Code:

```
gram = {"E":["E+T","T"],"T":["T*F","F"],"F":["(E)","i"],"S":["CC"],"C":["eC","d"],}
```

```
def removeDirectLR(gramA, A):
    temp,tempCr,tempInCr = gramA[A],[[],[]]
    for i in temp:
        if i[0] == A:
            tempInCr.append(i[1:]+[A+""])
        else:
            tempCr.append(i+[A+""])
    tempInCr.append(["e"])
    gramA[A],gramA[A+""] = tempCr,tempInCr
    return gramA
```

```
def checkForIndirect(gramA, a, ai):
    if ai not in gramA:
        return False
```

```

if a == ai:
    return True
for i in gramA[ai]:
    if i[0] == ai:
        return False
    if i[0] in gramA:
        return checkForIndirect(gramA, a, i[0])
return False

```

```

def rep(gramA, A):
    temp,newTemp = gramA[A],[]
    for i in temp:
        if checkForIndirect(gramA, A, i[0]):
            t = []
            for k in gramA[i[0]]:
                t=[]
                t+=k
                t+=i[1:]
                newTemp.append(t)
        else:
            newTemp.append(i)
    gramA[A] = newTemp
    return gramA

```

```

def rem(gram):
    c,conv,gramA,revconv = 1,{},{},{ }
    for j in gram:
        conv[j] = "A"+str(c)
        gramA["A"+str(c)] = []
        c+=1
    for i in gram:
        for j in gram[i]:
            temp = []
            for k in j:
                if k in conv:
                    temp.append(conv[k])
                else:
                    temp.append(k)
            gramA[conv[i]].append(temp)
    for i in range(c-1,0,-1):
        ai = "A"+str(i)
        for j in range(0,i):
            aj = gramA[ai][0][0]

```

```

        if ai!=aj :
            if aj in gramA and checkForIndirect(gramA,ai,aj):
                gramA = rep(gramA, ai)
    for i in range(1,c):
        ai = "A"+str(i)
        for j in gramA[ai]:
            if ai==j[0]:
                gramA = removeDirectLR(gramA, ai)
                break
    op = {}
    for i in gramA:
        a = str(i)
        for j in conv:
            a = a.replace(conv[j],j)
        revconv[i] = a
    for i in gramA:
        l = []
        for j in gramA[i]:
            k = []
            for m in j:
                if m in revconv:
                    k.append(m.replace(m,revconv[m]))
                else:
                    k.append(m)
            l.append(k)
        op[revconv[i]] = l
    return op

```

```

result,terminals = rem(gram),[]
for i in result:
    for j in result[i]:
        for k in j:
            if k not in result:
                terminals+= [k]
terminals = list(set(terminals))

```

```

def first(gram, term):
    a = []
    if term not in gram:
        return [term]
    for i in gram[term]:
        if i[0] not in gram:
            a.append(i[0])

```

```

        elif i[0] in gram:
            a += first(gram, i[0])
    return a

```

```

firsts = {}
for i in result:
    firsts[i] = first(result,i)

```

```

def follow(gram, term):
    a = []
    for rule in gram:
        for i in gram[rule]:
            if term in i:
                temp,indx = i,i.index(term)
                if indx+1!=len(i):
                    if i[-1] in firsts:
                        a+=firsts[i[-1]]
                    else:
                        a+=[i[-1]]
                else:
                    a+=["e"]
            if rule != term and "e" in a:
                a+= follow(gram,rule)
    return a

```

```

follows = {}
for i in result:
    follows[i] = list(set(follow(result,i)))
    if "e" in follows[i]:
        follows[i].pop(follows[i].index("e"))
    follows[i]+=["$"]

```

```

resMod = {}
for i in result:
    l = []
    for j in result[i]:
        temp = ""
        for k in j:
            temp+=k
        l.append(temp)
    resMod[i] = l
tterm = list(terminals)
tterm.pop(tterm.index("e"))

```

```

tterm+=["d"]
pptable = {}
for i in result:
    for j in tterm:
        if j in firsts[i]:
            pptable[(i,j)]=resMod[i[0]][0]
        else:
            pptable[(i,j)]=""
    if "e" in firsts[i]:
        for j in tterm:
            if j in follows[i]:
                pptable[(i,j)]="e"
pptable[("F","i")] = "i"
toprint = f'{"": <10}'
for i in tterm:
    toprint+= f' | {i: <10}'
print(toprint)
for i in result:
    toprint = f'{i: <10}'
    for j in tterm:
        if pptable[(i,j)]!="":
            toprint+=f' | {i+"->" +pptable[(i,j)]: <10}'
        else:
            toprint+=f' | {pptable[(i,j)]: <10}'
    print(f'{" "-: <76}')
    print(toprint)

```

```

1 gram = {"E":["E+T","T"],"T":["T*F","F"],"F":["(E)","i"],"S":["CC"],"C":["eC","d"],}
2
3 def removeDirectLR(gramA, A):
4     temp,tempCr,tempInCr = gramA[A],[],[]
5     for i in temp:
6         if i[0] == A:
7             tempInCr.append(i[1:]+[A+""])
8         else:
9             tempCr.append(i+[A+""])
10    tempInCr.append(["e"])
11    gramA[A],gramA[A+""] = tempCr,tempInCr
12    return gramA
13
14 def checkForIndirect(gramA, a, ai):
15     if ai not in gramA:
16         return False
17     if a == ai:
18         return True
19     for i in gramA[ai]:
20         if i[0] == ai:
21             return False
22         if i[0] in gramA:
23             return checkForIndirect(gramA, a, i[0])
24     return False
25
26 def rep(gramA, A):
27     temp,newTemp = gramA[A],[]
28     for i in temp:
29         if checkForIndirect(gramA, A, i[0]):
30             t = []
31             for k in gramA[i[0]]:
32                 t=[]
33                 t+=k
34                 t+=i[1:]
35                 newTemp.append(t)
36

```

```

37         newTemp.append(i)
38     gramA[A] = newTemp
39     return gramA
40
41 def rem(gram):
42     c,conv,gramA,revconv = 1,{},{},{}
43     for j in gram:
44         conv[j] = "A"+str(c)
45         gramA["A"+str(c)] = []
46         c+=1
47     for i in gram:
48         for j in gram[i]:
49             temp = []
50             for k in j:
51                 if k in conv:
52                     temp.append(conv[k])
53                 else:
54                     temp.append(k)
55             gramA[conv[i]].append(temp)
56     for i in range(c-1,0,-1):
57         ai = "A"+str(i)
58         for j in range(0,i):
59             aj = gramA[ai][0][0]
60             if ai!=aj :
61                 if aj in gramA and checkForIndirect(gramA,ai,aj):
62                     gramA = rep(gramA, ai)
63     for i in range(1,c):
64         ai = "A"+str(i)
65         for j in gramA[ai]:
66             if ai==j[0]:
67                 gramA = removeDirectLR(gramA, ai)
68                 break
69     op = {}
70     for i in gramA:
71         a = str(i)
72         for j in conv:

```

```

73         a = a.replace(conv[j],j)
74         revconv[i] = a
75     for i in gramA:
76         l = []
77         for j in gramA[i]:
78             k = []
79             for m in j:
80                 if m in revconv:
81                     k.append(m.replace(m,revconv[m]))
82                 else:
83                     k.append(m)
84             l.append(k)
85         op[revconv[i]] = l
86     return op
87
88 result,terminals = rem(gram),[]
89 for i in result:
90     for j in result[i]:
91         for k in j:
92             if k not in result:
93                 terminals+=k
94 terminals = list(set(terminals))
95
96 def first(gram, term):
97     a = []
98     if term not in gram:
99         return [term]
100    for i in gram[term]:
101        if i[0] not in gram:
102            a.append(i[0])
103        elif i[0] in gram:
104            a += first(gram, i[0])
105    return a
106
107 firsts = {}
108 for i in result:
109     firsts[i] = first(result,i)
110
111 def follow(gram, term):
112     a = []
113     for rule in gram:
114         for i in gram[rule]:
115             if term in i:
116                 temp,indx = i,i.index(term)
117                 if indx+1!=len(i):
118                     if i[-1] in firsts:
119                         a+=firsts[i[-1]]
120                     else:
121                         a+=i[-1]
122             else:
123                 a+=["e"]
124             if rule != term and "e" in a:
125                 a+= follow(gram,rule)
126     return a
127
128 follows = {}
129 for i in result:
130     follows[i] = list(set(follow(result,i)))
131     if "e" in follows[i]:
132         follows[i].pop(follows[i].index("e"))
133     follows[i]+=["$"]
134
135 resMod = {}
136 for i in result:
137     l = []
138     for j in result[i]:
139         temp = ""
140         for k in j:
141             temp+=k
142         l.append(temp)
143     resMod[i] = l
144 tterm = list(terminals)
145 tterm.pop(tterm.index("e"))
146 tterm+=["d"]
147 pptable = {}
148 for i in result:
149     for j in tterm:
150         if j in firsts[i]:
151             pptable[(i,j)]=resMod[i][0][0]
152         else:
153             pptable[(i,j)]=-""
154     if "e" in firsts[i]:
155         for j in tterm:
156             if j in follows[i]:
157                 pptable[(i,j)]= "e"
158 pptable[("F","i")] = "i"
159 toprint = f'{"": <10}'
160 for i in tterm:
161     toprint+= f'|{i: <10}'
162 print(toprint)
163 for i in result:
164     toprint = f'{i: <10}'
165     for j in tterm:
166         if pptable[(i,j)]!="":
167             toprint+=f'|{i+"-" +pptable[(i,j)]: <10}'
168         else:
169             toprint+=f'|{pptable[(i,j)]: <10}'
170     print(f'{"-": <76}')
171     print(toprint)

```

Output:

```
-----|d      |+      |i      |*      |)      |(      |d
E      |      |      |E->TE'|      |      |E->TE'|
-----|
T      |      |      |T->FT'|      |      |T->FT'|
-----|
F      |      |      |F->i  |      |      |F->(E)|
-----|
S      |S->CC|      |      |      |      |      |S->CC
-----|
C      |C->e  |      |      |      |      |      |C->e
-----|
E'     |      |E'->TE'|      |      |E'->e  |      |
-----|
T'     |      |T'->e  |      |T'->FT'|      |T'->e  |
-----|

...Program finished with exit code 0
Press ENTER to exit console.□
```

Result:

A program for Predictive parsing table was run successfully.