EXP 9: LR(0) Parsing

AIM:

To perform LR(0) parsing using python.

Algorithm:

Initialize the stack with the start state.

2. Read an input symbol

3. while true do

3.1 Using the top of the stack and the input symbol determine the next state.

3.2 If the next state is a stack state then

3.2.1 stack the state

3.2.2 get the next input symbol

3.3 else if the next state is a reduce state, then

3.3.1 output reduction number, k

3.3.2 pop RHS k -1 states from the stack where RHS k is the right-hand side of production k.

3.3.3 set the next input symbol to the LHS k

3.4 else if the next state is an accept state then

3.4.1 output valid sentence

3.4.2 return else

3.4.3 output invalid sentence

3.4.4 return

Source Code:

gram = {  
 "S":["CC"],  
 "C":["aC","d"]  
}  
start = "S"  
terms = ["a","d","$"]  
  
non\_terms = []  
*for* i *in* gram:  
 non\_terms.append(i)  
gram["S'"]= [start]  
  
  
new\_row = {}  
*for* i *in* terms+non\_terms:  
 new\_row[i]=""  
  
  
non\_terms += ["S'"]  
# each row in state table will be dictionary {nonterms ,term,$}  
stateTable = []  
# I = [(terminal, closure)]  
# I = [("S","A.A")]  
  
*def* Closure(*term*, *I*):  
 *if term in* non\_terms:  
 *for* i *in* gram[*term*]:  
 *I*+=[(*term*,"."+i)]  
 I = list(set(*I*))  
 *for* i *in I*:  
 # print("." != i[1][-1],i[1][i[1].index(".")+1])  
 *if* "." != i[1][-1] *and* i[1][i[1].index(".")+1] *in* non\_terms *and* i[1][i[1].index(".")+1] != *term*:  
 *I* += Closure(i[1][i[1].index(".")+1], [])  
 *return I*Is = []  
Is+=set(Closure("S'", []))  
  
  
countI = 0  
omegaList = [set(Is)]  
*while* countI<len(omegaList):  
 newrow = dict(new\_row)  
 vars\_in\_I = []  
 Is = omegaList[countI]  
 countI+=1  
 *for* i *in* Is:  
 *if* i[1][-1]!=".":  
 indx = i[1].index(".")  
 vars\_in\_I+=[i[1][indx+1]]  
 vars\_in\_I = list(set(vars\_in\_I))  
 # print(vars\_in\_I)  
 *for* i *in* vars\_in\_I:  
 In = []  
 *for* j *in* Is:  
 *if* "."+i *in* j[1]:  
 rep = j[1].replace("."+i,i+".")  
 In+=[(j[0],rep)]  
 *if* (In[0][1][-1]!="."):  
 temp = set(Closure(i,In))  
 *if* temp *not in* omegaList:  
 omegaList.append(temp)  
 *if* i *in* non\_terms:  
 newrow[i] = str(omegaList.index(temp))  
 *else*:  
 newrow[i] = "s"+str(omegaList.index(temp))  
 print(f'Goto(I{countI-1},{i}):{temp} That is I{omegaList.index(temp)}')  
 *else*:  
 temp = set(In)  
 *if* temp *not in* omegaList:  
 omegaList.append(temp)  
 *if* i *in* non\_terms:  
 newrow[i] = str(omegaList.index(temp))  
 *else*:  
 newrow[i] = "s"+str(omegaList.index(temp))  
 print(f'Goto(I{countI-1},{i}):{temp} That is I{omegaList.index(temp)}')  
  
 stateTable.append(newrow)  
print("\n\nList of I's\n")  
*for* i *in* omegaList:  
 print(f'I{omegaList.index(i)}: {i}')  
  
  
#populate replace elements in state Table  
I0 = []  
*for* i *in* list(omegaList[0]):  
 I0 += [i[1].replace(".","")]  
print(I0)  
  
*for* i *in* omegaList:  
 *for* j *in* i:  
 *if* "." *in* j[1][-1]:  
 *if* j[1][-2]=="S":  
 stateTable[omegaList.index(i)]["$"] = "Accept"  
 *break  
 for* k *in* terms:  
 stateTable[omegaList.index(i)][k] = "r"+str(I0.index(j[1].replace(".","")))  
print("\nStateTable")  
  
print(f'{" ": <9}',end="")  
*for* i *in* new\_row:  
 print(f'|{i: <11}',end="")  
  
print(f'\n{"-":-<66}')  
*for* i *in* stateTable:  
 print(f'{"I("+str(stateTable.index(i))+")": <9}',end="")  
 *for* j *in* i:  
 print(f'|{i[j]: <10}',end=" ")  
 print()

Output:

C:\Users\hp\AppData\Local\Programs\Python\Python39\python.exe "F:/Python/DAA/Compiler Design/lrp.py"

Goto(I0,S):{("S'", 'S.')} That is I1

Goto(I0,a):{('C', '.d'), ('C', 'a.C'), ('C', '.aC')} That is I2

Goto(I0,d):{('C', 'd.')} That is I3

Goto(I0,C):{('S', 'C.C'), ('C', '.d'), ('C', '.aC')} That is I4

Goto(I2,a):{('C', '.d'), ('C', 'a.C'), ('C', '.aC')} That is I2

Goto(I2,d):{('C', 'd.')} That is I3

Goto(I2,C):{('C', 'aC.')} That is I5

Goto(I4,a):{('C', '.d'), ('C', 'a.C'), ('C', '.aC')} That is I2

Goto(I4,d):{('C', 'd.')} That is I3

Goto(I4,C):{('S', 'CC.')} That is I6

List of I's

I0: {('S', '.CC'), ('C', '.d'), ("S'", '.S'), ('C', '.aC')}

I1: {("S'", 'S.')}

I2: {('C', '.d'), ('C', 'a.C'), ('C', '.aC')}

I3: {('C', 'd.')}

I4: {('S', 'C.C'), ('C', '.d'), ('C', '.aC')}

I5: {('C', 'aC.')}

I6: {('S', 'CC.')}

['CC', 'd', 'S', 'aC']

StateTable

|a |d |$ |S |C

------------------------------------------------------------------

I(0) |s2 |s3 | |1 |4

I(1) | | |Accept | |

I(2) |s2 |s3 | | |5

I(3) |r1 |r1 |r1 | |

I(4) |s2 |s3 | | |6

I(5) |r3 |r3 |r3 | |

I(6) |r0 |r0 |r0 | |

Process finished with exit code 0

Result:

Hence the LR(0) parsing is performed successfully.