Harshit Aggarwal (RA1911003010782)

Experiment 9: Implementation of LR(0) Parsing

Aim:
To implement LR(0) Parsing in code.

Algorithm:
1. Start
2. Writing augmented grammar
3. LR(0) collection of items to be found
4. Defining 2 functions: goto(list of terminals) and action(list of non-terminals) in the parsing table.
5. Stop

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]):
```

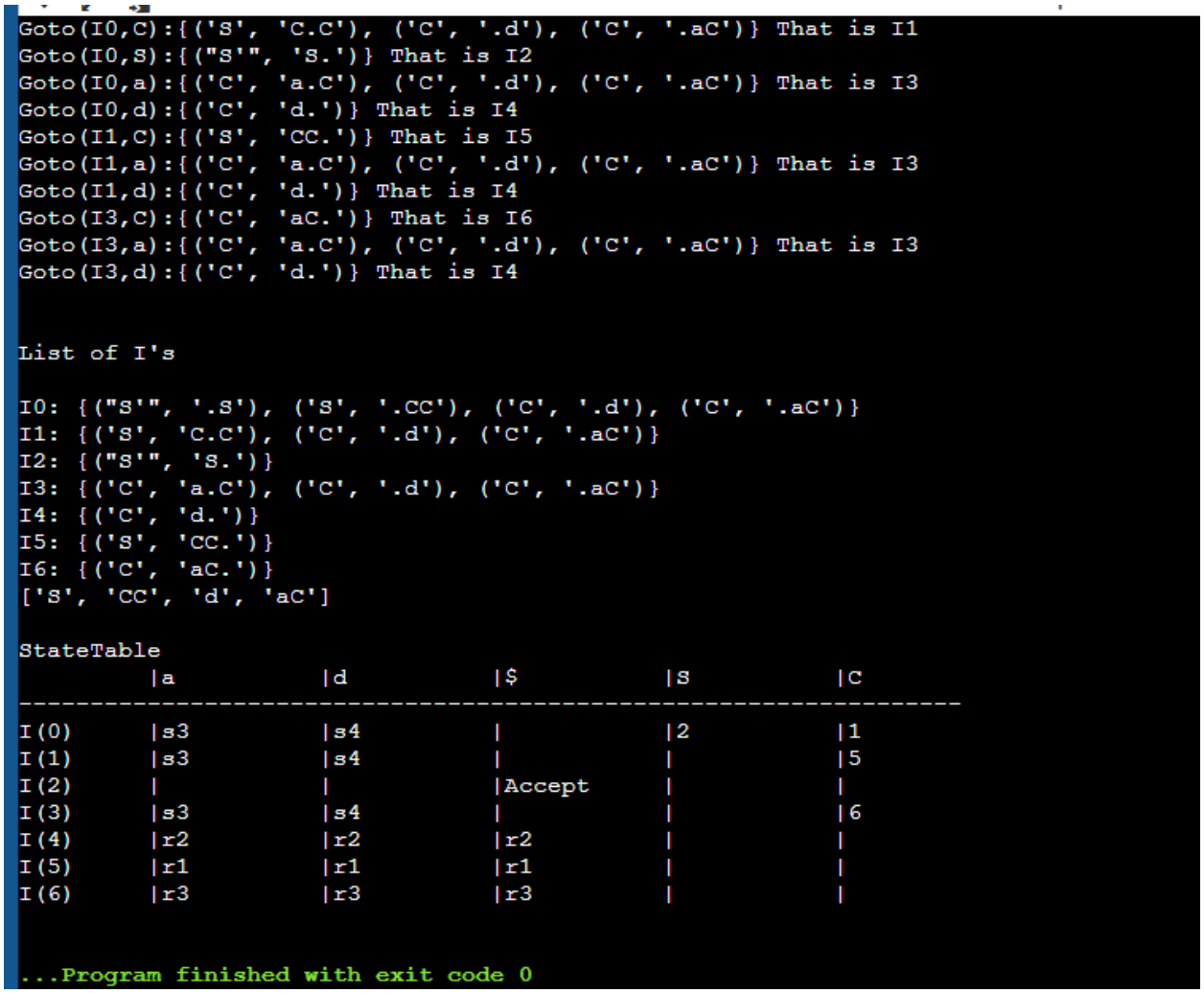
```
        I0 += [i[1].replace(".","")]
print(I0)


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:



```
Goto(I0,C):{('S', 'C.C'), ('C', '.d'), ('C', '.aC')} That is I1
Goto(I0,S):{("S'", 'S.')} That is I2
Goto(I0,a):{('C', 'a.C'), ('C', '.d'), ('C', '.aC')} That is I3
Goto(I0,d):{('C', 'd.')} That is I4
Goto(I1,C):{('S', 'CC.')} That is I5
Goto(I1,a):{('C', 'a.C'), ('C', '.d'), ('C', '.aC')} That is I3
Goto(I1,d):{('C', 'd.')} That is I4
Goto(I3,C):{('C', 'aC.')} That is I6
Goto(I3,a):{('C', 'a.C'), ('C', '.d'), ('C', '.aC')} That is I3
Goto(I3,d):{('C', 'd.')} That is I4


List of I's

I0: {("S'", '.S'), ('S', '.CC'), ('C', '.d'), ('C', '.aC')}
I1: {('S', 'C.C'), ('C', '.d'), ('C', '.aC')}
I2: {("S'", 'S.')}
I3: {('C', 'a.C'), ('C', '.d'), ('C', '.aC')}
I4: {('C', 'd.')}
I5: {('S', 'CC.')}
I6: {('C', 'aC.')}
['S', 'CC', 'd', 'aC']

StateTable
         |a          |d          |$          |S          |C
-------------------------------------------------------------------
I(0)     |s3         |s4         |           |2          |1
I(1)     |s3         |s4         |           |           |5
I(2)     |           |           |Accept     |           |
I(3)     |s3         |s4         |           |           |6
I(4)     |r2         |r2         |r2         |           |
I(5)     |r1         |r1         |r1         |           |
I(6)     |r3         |r3         |r3         |           |


...Program finished with exit code 0
```

Result:

Hence LR(0) parsing was done for the given grammar.