**NAME –** DEVANGI PURKAYASTHA

**REGISTRATION NUMBER –** RA1811003010689

**SECTION-** D2

LAB EXP:9 COMPUTATION OF LR(0) ITEMS

DATE: 26 March 2021

**AIM : To Compute LR(0) ITEMS**

**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()

**ALGORITHM-**

* The LR Parser is a Shift-reduce Parser that makes use of a Deterministic Finite Automata, recognizing the Set Of All Viable Prefixes by reading the stack from Bottom To Top.
* If a Finite-State Machine that recognizes viable prefixes of the right sentential forms is constructed, it can be used to guide the handle selection in the Shift-reduce Parser.
* Handle: Handle is a substring that matches the body of a production
* Handle is a Right Sentential Form + position where reduction can be performed + production used for reduction.

**LR(0) Items**

-An LR(0) Item of a Grammar G is a Production of G with a Dot (.) at some position of the right side. Production A → XYZ yields the Four items:

1. A→•XYZ We hope to see a string derivable from XYZ next on the input.

2. A→X•YZ We have just seen on the input a string derivable from X and that we hope next to see a string derivable from YZ next on the input.

3. A→XY•Z

4. A→XYZ•

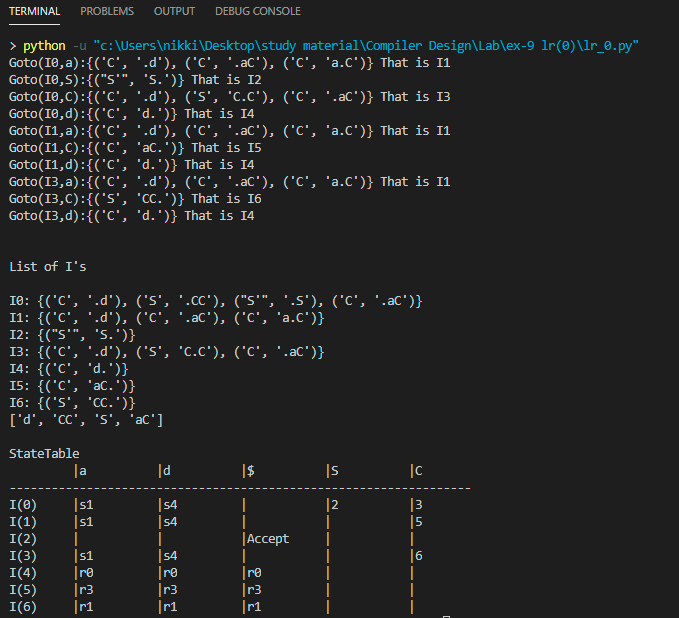
❖ The production A→ε generates only one item, A→•.

❖ Each of this item is a Viable prefixes

❖ Closure Item : An Item created by the closure operation on a state.

❖ Complete Item : An Item where the Item Dot is at the end of the RHS.

**OUTPUT** –



**RESULT –**

The given program has been successfully executed.