Exp:9: Computation of LR (0) Items

AIM: To design a code to computation of LR (0) items.

LANGUAGE USED: Python 3

ALGORITHM/PROCEDURE: -

- Epsilon is represented by ‘e’.
- Productions are of the form A=B, where ‘A’ is a single Non-Terminal and ‘B’ can be any combination of Terminals and Non- Terminals.
- Each production of a non-terminal is entered on a different line.
- Only Upper-Case letters are Non-Terminals and everything else is a terminal.
- Do not use ‘!’ or ‘$’ as they are reserved for special purposes.
- Grammar is taken as input and will be through an infinite loop (while=true).

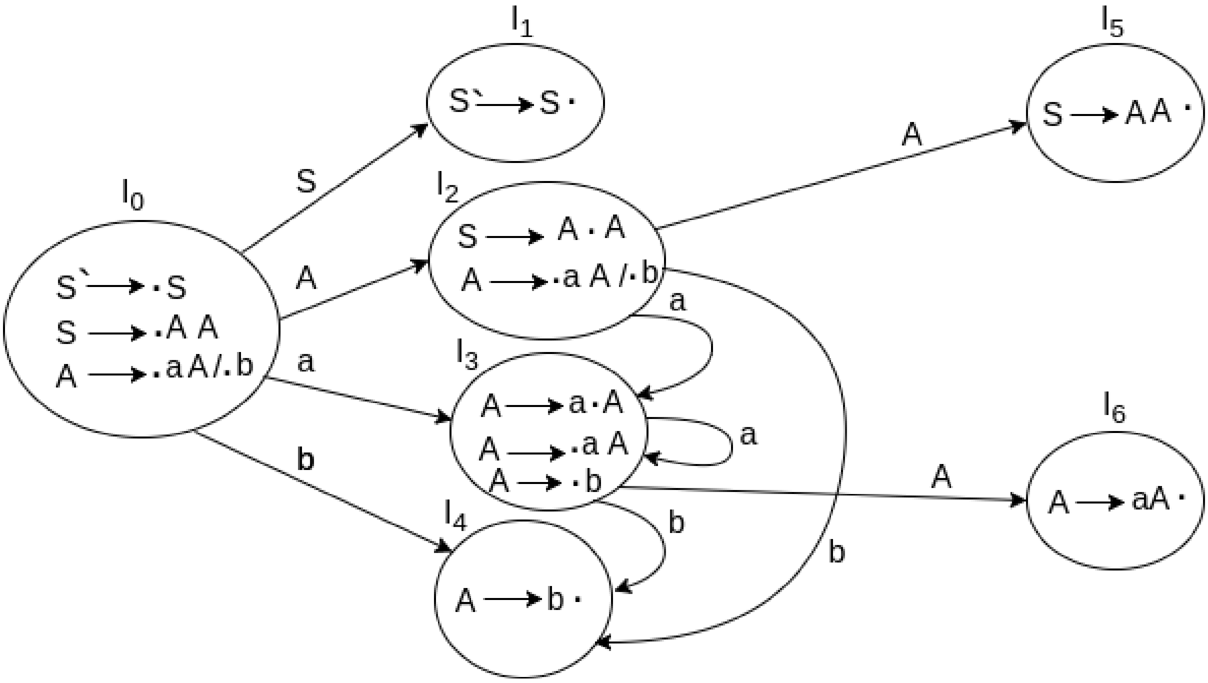
EXPLANATION:

An LR (0) item is a production of the grammar with exactly one dot on the right-hand side. For example, production T → T * F leads to four LR (0) items:

T → · T * F
T → T · * F
T → T * · F
T → T * F ·
What is to the left of the dot has just been read, and the parser is ready to read the remainder, after the dot.

Two LR (0) items that come from the same production but have the dot in different places are considered different LR (0) items.

SPACE TREE DIAGRAM/ EXPLANATION:

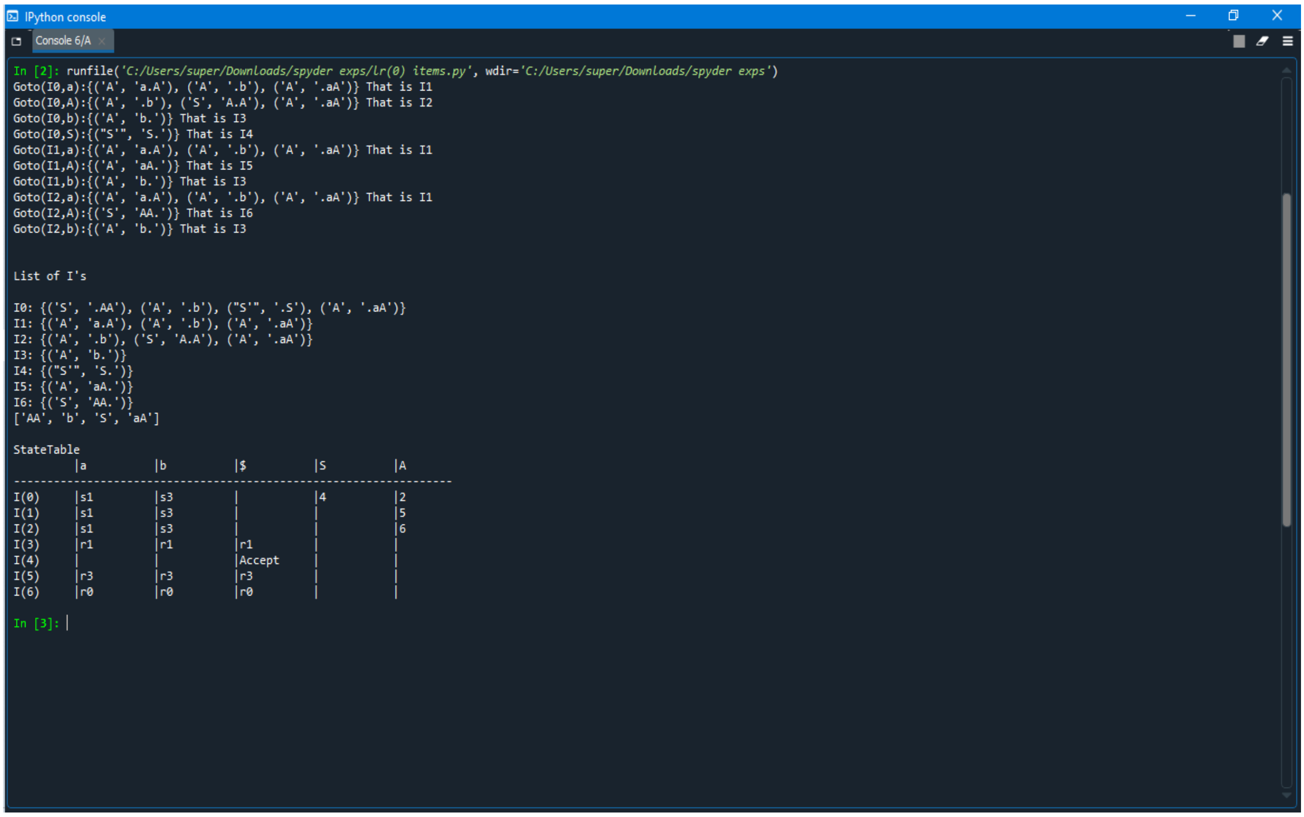


SOURCE CODE: -

```
gram = {"S":["AA"],
        "A":["aA","b"]
}

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'"]

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:



```
In [2]: runfile('C:/Users/super/Downloads/spyder exps/lr(0) items.py', wdir='C:/Users/super/Downloads/spyder exps')
Goto(I0,a):{('A', 'a.A'), ('A', '.b'), ('A', '.aA')} That is I1
Goto(I0,A):{('A', '.b'), ('S', 'A.A'), ('A', '.aA')} That is I2
Goto(I0,b):{('A', 'b.')} That is I3
Goto(I0,S):{("S'", 'S.')} That is I4
Goto(I1,a):{('A', 'a.A'), ('A', '.b'), ('A', '.aA')} That is I1
Goto(I1,A):{('A', 'aA.')} That is I5
Goto(I1,b):{('A', 'b.')} That is I3
Goto(I2,a):{('A', 'a.A'), ('A', '.b'), ('A', '.aA')} That is I1
Goto(I2,A):{('S', 'AA.')} That is I6
Goto(I2,b):{('A', 'b.')} That is I3


List of I's

I0: {('S', '.AA'), ('A', '.b'), ("S'", '.S'), ('A', '.aA')}
I1: {('A', 'a.A'), ('A', '.b'), ('A', '.aA')}
I2: {('A', '.b'), ('S', 'A.A'), ('A', '.aA')}
I3: {('A', 'b.')}
I4: {("S'", 'S.')}
I5: {('A', 'aA.')}
I6: {('S', 'AA.')}
['AA', 'b', 'S', 'aA']

StateTable
         |a          |b          |$          |S          |A
------------------------------------------------------------------
I(0)     |s1         |s3         |           |4          |2
I(1)     |s1         |s3         |           |           |5
I(2)     |s1         |s3         |           |           |6
I(3)     |r1         |r1         |r1         |           |
I(4)     |           |           |Accept     |           |
I(5)     |r3         |r3         |r3         |           |
I(6)     |r0         |r0         |r0         |           |

In [3]:
```

RESULT: Therefore, we successfully implemented a code for computing LR (0) items for given grammar.