SS ASSIGNMENT -06

ROLL NO: U21CS052
NAME : PANCHAL GUNGUN PARESH

LR(1) PARSER

first.py

```
def first(grammar):
    # first & follow sets, epsilon-productions


    first = {i: set() for i in grammar.nonterminals}
    first.update((i, {i}) for i in grammar.terminals)
    epsilon=set()


    while True:
        updated = False


        for nt, expression in grammar.rules:
            # FIRST set w.r.t epsilon-productions
            for symbol in expression:
                updated |= union(first[nt], first[symbol])
                if symbol not in epsilon:
                    break
            else:
```

```
                updated |= union(epsilon, {nt})



        if not updated:
            return first



def union(first, begins):
    n = len(first)
    first |= begins
    return len(first) != n



class Grammar:

    def __init__(self, *rules):
        self.rules = tuple(self._parse(rule) for rule in rules)

    def _parse(self, rule):
        return tuple(rule.replace(' ', '').split('::='))

    def __getitem__(self, nonterminal):
        yield from [rule for rule in self.rules
                    if rule[0] == nonterminal]

    @staticmethod
    def is_nonterminal(symbol):
```

```
        return symbol.isalpha() and symbol.isupper()


    @property
    def nonterminals(self):
        return set(nt for nt, _ in self.rules)


    @property
    def terminals(self):
        return set(
            symbol
            for _, expression in self.rules
            for symbol in expression
            if not self.is_nonterminal(symbol)
        )



first = first(Grammar(
    '^::=S$',
    'S::=CC',
    'C::=cC',
    'C::=d',
)
)
```

items.py

```
''''

Find Closure and Contents of a CLR Parsing Table

input>>
    '^::=S$',
    'S::=CC',
    'C::=cC',
    'C::=d'

Output>>
    {'0+S': 'shift 2', '0+C': 'shift 3', '0+c': 'shift 10', '0+d': 'shift
8',
     '1+d': 'reduce C::=d', '1+c': 'reduce C::=d', '2+$': 'reduce ^::=S',
'3+C': 'shift 6',
     '3+c': 'shift 7', '3+d': 'shift 5', '4+C': 'shift 12', '4+c': 'shift
10', '4+d': 'shift 8',
    '5+$': 'reduce C::=d', '6+$': 'reduce S::=CC', '7+C': 'shift 11',
'7+c': 'shift 7',
    '7+d': 'shift 5', '8+d': 'reduce C::=d', '9+d': 'reduce C::=cC',
'9+c': 'reduce C::=cC',
    '10+C': 'shift 12', '10+c': 'shift 10', '10+d': 'shift 8', '11+$':
'reduce C::=cC', '12+d': 'reduce C::=cC'}
    (Contents Of Parsing Table)
    where an entry,
    5+$: 'reduce C::=d' represents reduction in state 5 , under $ symbol
from d to C.


    State  0
    [['^::=.S$'], ['S::=.CC$'], ['C::=.cCd'], ['C::=.cCc'], ['C::=.dd'],
['C::=.dc']]
    **********
    State  1
    [['C::=d.d'], ['C::=d.c']]
    **********
```

```
State  2
[['^::=S.$']]
**********
State  3
[['S::=C.C$'], ['C::=.cC$'], ['C::=.d$']]
**********
State  4
[['C::=c.Cd'], ['C::=c.Cc'], ['C::=.cCd'], ['C::=.dd']]
**********
State  5
[['C::=d.$']]
**********
State  6
[['S::=CC.$']]
**********
State  7
[['C::=c.C$'], ['C::=.cC$'], ['C::=.d$']]
**********
State  8
[['C::=d.d']]
**********
State  9
[['C::=cC.d'], ['C::=cC.c']]
**********
State  10
[['C::=c.Cd'], ['C::=.cCd'], ['C::=.dd']]
**********
State  11
[['C::=cC.$']]
**********
State  12
[['C::=cC.d']]
**********


All Possible States
```

```
'''









from first import first

def checkValidity(i):
    if i[0][-1]=='.':
        return False
    return True


def preProcessStates(states):
    '''
    One of the implementation error:
    Since our grammar is in the form 'A::=Bb' where b is the follow of the grammar.

    The program could not distinguish the follow element from the other elements, hence, we need this function to
    avoid that.

    '''
    l=[]
    for i in states:
        if checkValidity(i):
            l.append(''.join(i).replace(' ',''))

    if len(l)!=0:
        return(l)


def is_nonterminal(symbol):
    return symbol.isupper()
```

```
def shiftPos(item):
    Item=''.join(item).replace(' ','')
    listItem=list(Item)
    index=listItem.index('.')
    if len(listItem[index:])!=1:
        return (Item[:index]+''+Item[index+1]+'.'+Item[index+2:])
    return item
def check(item,N):
    '''
    Check if the grammar is not completely parsed or not
    Input: Item, GrammarSymbol
    Output: True if . can be shifted else False

    Example1:
    Input:['A::=B.b$'],b
    Output:True

    '''
    Item=''.join(item).replace(' ','')
    listItem=list(Item)
    try:
        index=listItem.index('.')
        #index1=listItem.index('=')
        if N ==listItem[index+1]:
            return True
        if ' '== listItem[index+1]:
            return False
    except:
        return False
def GOTO(I,N):
    '''
    Input: (Item,GrammarSymbol)
    Output: Closure of Item after shift grammar GrammarSymbol is performed

    Example:
    Input: ['S::=.CC$'],C
    Output: [['S::=C.C$'], ['C::=.cC$'], ['C::=.d$']]

    '''
    J=[]
```

```
    for i in I:
        if check(i,N):
            new=shiftPos(i)
            J.append(new)

    if len(J)==0:
        return([])

    return(findClosure([J]))

def allGrammarSymbol(item):
    '''
    Input: All sets of Grammar(our main input)
    Output: Grammar Symbols
    '''
    l=[]
    for i in item:
        for k in i:
            if k.isalpha():
                l.append(k)

    return set(l)


def findProduction(B):
    '''
    Input: A non-terminal B
    Output: All Productions of B
    Example
    Input : 'S'
    Output: ['CC']


    '''

    if B=='$':
        return 1
    if B not in entryOfGram.keys():
        return 1
    return entryOfGram[B]
```

```
def findTerminalsOf(gram):
    newList={}
    for i in gram:
        n=i.replace(' ', '').split('::=')
        if n[0] not in newList.keys():
            newList[n[0]]=[''.join(n[1])]
        else:
            newList[n[0]].append(''.join(n[1]))
    return newList


def nextDotPos(item):
    '''
    input: An item
    output: The element to be executed

    Example
    input:'A::=.A$'
    output :'A'

    '''
    Item=item.replace(' ','')
    listItem=list(Item)
    try:
        index=listItem.index('.')
        return listItem[index+1]
    except:
        return '$'

def followOf(item):
    '''
    input: An item
    output: The next non-terminal to be opened up

    Example
    input:'A::.AB'
    output: 'B'

    '''
```

```
    Item=item.replace(' ','')
    listItem=list(Item)
    try:
        index=listItem.index('.')
        return listItem[index+2]
    except IndexError:
        return '$'

def findClosure(I):
    '''
    Input: Grammar I
    Output : Closure of Grammar

    Input: '^::=.S$''
    Output : [['^::=.S$'], ['S::=.CC$'], ['C::=.cCd'], ['C::=.cCc'],
['C::=.dd'], ['C::=.dc']]

    where last element is the follow element.
    Example:
    In ['^::=.S$'] , $ is the follow of '^::=.S'


    '''
    add=1
    while (add!=0):
        add=0
        for item in I:
            element=item[0]
            giveElement=nextDotPos(element)
            findPr=findProduction(giveElement)
            if findPr==1:
                pass
            else:
                for productions in findPr:
                    for b in first[followOf(element)]:
                        elem=[giveElement+'::=.'+productions+''+b]
                        if elem not in I:
                            I.append(elem)
                            add=1
        return(I)
```

```
        break

gram=(
    '^::=S$',
    'S::=CC',
    'C::=cC',
    'C::=d'
)

# You can update your Grammar here. Be sure you update it on first.py line
no. 61 as well.
# Also add the augmented Grammar like in line 269
# Adjust line 278 acc. to your need

starting='^::=.S$'

entryOfGram=findTerminalsOf(gram)
I=[findClosure([[starting]])]
#findClosure(GOTO(I[0],'d'))

X=allGrammarSymbol(gram)

allItems={}
ItemsAll=[]
new_item=True
while new_item:
    new_item=False
    i=1
    for item in I:
        i+=1
        for g in X:
            if len(GOTO(item,g))!=0:
                goto=GOTO(item,g)
                flat_list = [[item] for sublist in goto for item in
sublist]
                if flat_list not in I:
                    index='I'+str(i)
                    if index not in allItems.keys():
                        allItems[index]=[g]
                    else:
```

```
                        allItems[index].append(g)
                    I.append(flat_list)
                    Z=preProcessStates(flat_list)
                    if (Z):
                        ItemsAll.append(flat_list)

                    new_item=True

    new_item=False

ItemsAll.insert(0,findClosure([[starting]]))
i=0
ACTION={}
#print(ItemsAll)
print('********************')
for item in ItemsAll:
    i+=1
    for num in item:
        x=list(num[0]).index('.')+2
        y=len(num[0])
        if x<y:
            elem=list(num[0]).index('.')+1
            gotoElem=list(num[0])[elem]
            IJ= GOTO(num,gotoElem)
            #print(IJ)
            if IJ in ItemsAll:
                #print('aa')
                #print(num)
                index=ItemsAll.index(IJ)
                last=list(num[0])[len(num[0])-1]
                ACTION[str(i-1)+'+'+gotoElem]="shift "+str(index)
        else:
            listy=list(num[0]).index('.')
            el=num[0][listy+1]
            ACTION[str(i-1)+'+'+el]="reduce "+num[0][:listy]

        #print(num[0])


print(ACTION)
```

```
print('************************')



print('**** All States Are ****')
for i in ItemsAll:
    print('State ',ItemsAll.index(i))
    print(i)
    print('**********************')
```

```
Microsoft Windows [Version 10.0.22631.3296]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dell\Desktop\study\allStudyMaterial-\sem 6\04_ss\01_labs\lab06>C:/Users/Dell/AppData/Local/Programs/Python/Python311/pyth
on.exe "c:/Users/Dell/Desktop/study/allStudyMaterial-/sem 6/04_ss/01_labs/lab06/items.py"
**********
{'0+S': 'shift 2', '0+C': 'shift 4', '0+c': 'shift 5', '0+d': 'shift 6', '1+C': 'shift 11', '1+c': 'shift 5', '1+d': 'shift 6', '2
+$': 'reduce ^::=S', '3+c': 'reduce C::=d', '3+d': 'reduce C::=d', '4+C': 'shift 10', '4+c': 'shift 8', '4+d': 'shift 9', '5+C': '
shift 11', '5+c': 'shift 5', '5+d': 'shift 6', '6+c': 'reduce C::=d', '7+c': 'reduce C::=cC', '7+d': 'reduce C::=cC', '8+C': 'shif
t 12', '8+c': 'shift 8', '8+d': 'shift 9', '9+$': 'reduce C::=d', '10+$': 'reduce S::=CC', '11+c': 'reduce C::=cC', '12+$': 'reduc
e C::=cC'}
**********
**** All States Are ****
State  0
[['^::=.S$'], ['S::=.CC$'], ['C::=.cCc'], ['C::=.cCd'], ['C::=.dc'], ['C::=.dd']]
**********
State  1
[['C::=c.Cc'], ['C::=c.Cd'], ['C::=.cCc'], ['C::=.dc']]
**********
State  2
[['^::=S.$']]
**********
State  3
[['C::=d.c'], ['C::=d.d']]
**********
State  4
[['S::=C.C$'], ['C::=.cC$'], ['C::=.d$']]
**********
State  5
[['C::=c.Cc'], ['C::=.cCc'], ['C::=.dc']]
**********
```

```
**********
State  6
[['C::=d.c']]
**********
State  7
[['C::=cC.c'], ['C::=cC.d']]
**********
State  8
[['C::=c.C$'], ['C::=.cC$'], ['C::=.d$']]
**********
State  9
[['C::=d.$']]
**********
State  10
[['S::=CC.$']]
**********
State  11
[['C::=cC.c']]
**********
State  12
[['C::=cC.$']]
**********

C:\Users\Dell\Desktop\study\allStudyMaterial-\sem 6\04_ss\01_labs\lab06>
```