System Software
Assignment 9

U20CS005
BANSI MARAKANA

1. Write a program to construct LALR () parse table for the following grammar and check whether the given input can be accepted or not.
Grammar:
S -> AA
A -> aA
A -> b
firstfollow.py

```
from re import *
from collections import OrderedDict
t_list=OrderedDict()
nt_list=OrderedDict()
production_list=[]
class Terminal:
    def __init__(self, symbol):
        self.symbol=symbol
    def __str__(self):
        return self.symbol

class NonTerminal:
    def __init__(self, symbol):
        self.symbol=symbol
        self.first=set()
        self.follow=set()
    def __str__(self):
        return self.symbol
    def add_first(self, symbols): self.first |= set(symbols) #union operation
    def add_follow(self, symbols): self.follow |= set(symbols)

def compute_first(symbol): #chr(1013) corresponds (ϵ) in Unicode
    global production_list, nt_list, t_list
    if symbol in t_list:
        return set(symbol)
    for prod in production_list:
        head, body=prod.split('->')
        if head!=symbol: continue
        if body=='':
            nt_list[symbol].add_first(chr(1013))
            continue
        for i, Y in enumerate(body):
            if body[i]==symbol: continue
            t=compute_first(Y)
            nt_list[symbol].add_first(t-set(chr(1013)))
            if chr(1013) not in t:
                break
            if i==len(body)-1:
                nt_list[symbol].add_first(chr(1013))

    return nt_list[symbol].first

def get_first(symbol): #wrapper method for compute_first
    return compute_first(symbol)

def compute_follow(symbol):
    global production_list, nt_list, t_list
    if symbol == list(nt_list.keys())[0]: #this is okay since I'm using an OrderedDict
        nt_list[symbol].add_follow('$')
    for prod in production_list:
        head, body=prod.split('->')
        for i, B in enumerate(body):
            if B != symbol: continue
            if i != len(body)-1:
                nt_list[symbol].add_follow(get_first(body[i+1]) - set(chr(1013)))
            if i == len(body)-1 or chr(1013) in get_first(body[i+1]) and B != head:
                nt_list[symbol].add_follow(get_follow(head))

def get_follow(symbol):
    global nt_list, t_list
    if symbol in t_list.keys():
        return None
    return nt_list[symbol].follow

def main(pl=None):
    print('''Enter the grammar productions (enter 'end' or return to stop)
#(Format: "A->Y1Y2..Yn" {Yi - single char} OR "A->" {epsilon})''')
    global production_list, t_list, nt_list
    ctr=1
    #t_regex, nt_regex=r'[a-z\W]', r'[A-Z]'
    if pl==None:
        while True:
            #production_list.append(input('{})\t'.format(ctr)))
            production_list.append(input().replace(' ', ''))
            if production_list[-1].lower() in ['end', '']:
                del production_list[-1]
                break
            head, body=production_list[ctr-1].split('->')
            if head not in nt_list.keys():
                nt_list[head]=NonTerminal(head)
            #for all terminals in the body of the production
            for i in body:
                if not 65<=ord(i)<=90:
                    if i not in t_list.keys(): t_list[i]=Terminal(i)
            #for all non-terminals in the body of the production
                elif  i not in nt_list.keys(): nt_list[i]=NonTerminal(i)
            ctr+=1

    return pl,production_list

if __name__=='__main__':

    main()
```

LALR CODE

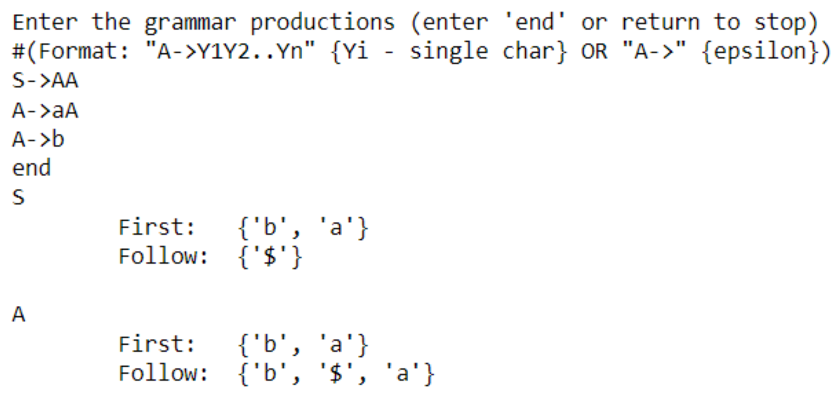
```
from graphviz import Digraph
from collections import OrderedDict
import firstfollow
from firstfollow import production_list, nt_list as ntl, t_list as tl
nt_list, t_list=[], []
dot = Digraph(comment='DFA for LALR')

class State:
    _id=0
    def __init__(self, closure):
        self.closure=closure
        self.no=State._id
        State._id+=1

class Item(str):
    def __new__(cls, item, lookahead=list()):
        self=str.__new__(cls, item)
        self.lookahead=lookahead
        return self
    def __str__(self):
        return super(Item, self).__str__()+", "+'|'.join(self.lookahead)
```

```
def closure(items):
    def exists(newitem, items):
        for i in items:
            if i==newitem and sorted(set(i.lookahead))==sorted(set(newitem.lookahead)):
                return True
        return False

    global production_list
    while True:
        flag=0
        for i in items:
            if i.index('.')==len(i)-1: continue
            Y=i.split('->')[1].split('.')[1][0]
            if i.index('.')+1<len(i)-1:
                lastr=list(firstfollow.compute_first(i[i.index('.')+2])-set(chr(1013)))
            else:
                lastr=i.lookahead
            for prod in production_list:
                head, body=prod.split('->')
                if head!=Y: continue
                newitem=Item(Y+'->.'+body, lastr)
                if not exists(newitem, items):
                    items.append(newitem)
                    flag=1
        if flag==0: break
    return items

def goto(items, symbol):
    dot.node(symbol,str(items))
    global production_list
    initial=[]
    for i in items:
        if i.index('.')==len(i)-1: continue
        head, body=i.split('->')
        seen, unseen=body.split('.')
        if unseen[0]==symbol and len(unseen) >= 1:
            initial.append(Item(head+'->'+seen+unseen[0]+'.'+unseen[1:], i.lookahead))
    return closure(initial)

def calc_states():
    def contains(states, t):
        for s in states:
            if len(s) != len(t): continue
            if sorted(s)==sorted(t):
                for i in range(len(s)):
                        if s[i].lookahead!=t[i].lookahead: break
                else: return True
        return False
    global production_list, nt_list, t_list
    head, body=production_list[0].split('->')
    states=[closure([Item(head+'->.'+body, ['$'])])]
    while True:
        flag=0
        for s in states:
            for e in nt_list+t_list:
                t=goto(s, e)
                if t == [] or contains(states, t): continue
                states.append(t)
                flag=1
        if not flag: break
    return states

def make_table(states):
    global nt_list, t_list
    def getstateno(t):
        for s in states:
            if len(s.closure) != len(t): continue
            if sorted(s.closure)==sorted(t):
                for i in range(len(s.closure)):
                        if s.closure[i].lookahead!=t[i].lookahead: break
                else: return s.no
        return -1

    def getprodno(closure):
        closure=''.join(closure).replace('.', '')
        return production_list.index(closure)
    SLR_Table=OrderedDict()
    for i in range(len(states)):
        states[i]=State(states[i])
    for s in states:
        SLR_Table[s.no]=OrderedDict()
        for item in s.closure:
            head, body=item.split('->')
            if body=='.':
                for term in item.lookahead:
                    if term not in SLR_Table[s.no].keys():
                        SLR_Table[s.no][term]={'r'+str(getprodno(item))}
                    else: SLR_Table[s.no][term] |= {'r'+str(getprodno(item))}
                continue
            nextsym=body.split('.')[1]
            if nextsym=='':
                if getprodno(item)==0:
                    SLR_Table[s.no]['$']='accept'
                else:
                    for term in item.lookahead:
                        if term not in SLR_Table[s.no].keys():
                            SLR_Table[s.no][term]={'r'+str(getprodno(item))}
                        else: SLR_Table[s.no][term] |= {'r'+str(getprodno(item))}
                continue
            nextsym=nextsym[0]
            t=goto(s.closure, nextsym)
            if t != []:
                if nextsym in t_list:
                    if nextsym not in SLR_Table[s.no].keys():
                        SLR_Table[s.no][nextsym]={'s'+str(getstateno(t))}
                    else: SLR_Table[s.no][nextsym] |= {'s'+str(getstateno(t))}
                else: SLR_Table[s.no][nextsym] = str(getstateno(t))
    return SLR_Table

def augment_grammar():
    for i in range(ord('Z'), ord('A')-1, -1):
        if chr(i) not in nt_list:
            start_prod=production_list[0]
            production_list.insert(0, chr(i)+'->'+start_prod.split('->')[0])
            return
pl,prod_list = firstfollow.main()
pro = prod_list.copy()
for nt in ntl:
    firstfollow.compute_first(nt)
    firstfollow.compute_follow(nt)
    print(nt)
    print("\tFirst:\t", firstfollow.get_first(nt))
    print("\tFollow:\t", firstfollow.get_follow(nt), "\n")
augment_grammar()
nt_list=list(ntl.keys())
t_list=list(tl.keys()) + ['$']
cs=calc_states()
items = []
ctr=0
```

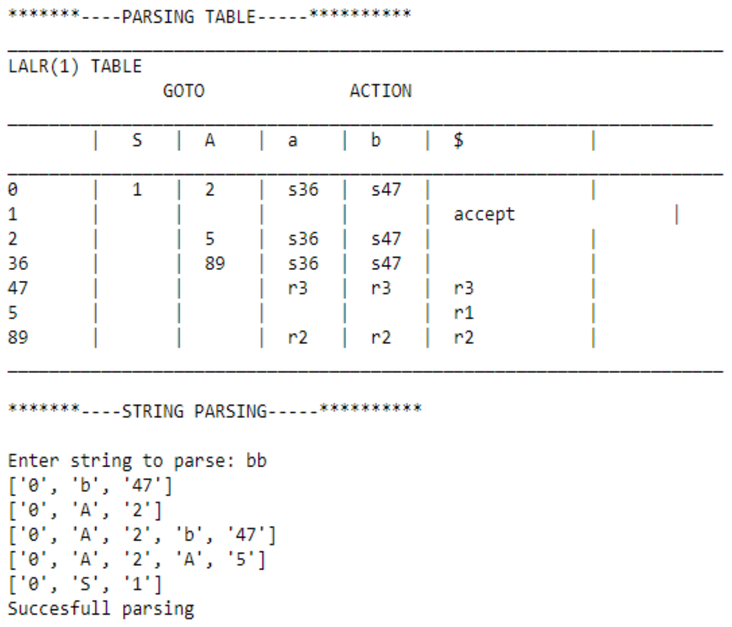
```
m = [ ]
for s in cs:
    items.append(str(ctr))
    ctr+=1
check = []
count = 0
ind = []
for i in cs:
    if i not in check:
        check.append(i)
    else:
        ind.append(count)
    count += 1
merge_ind = []
combine = []
for i in ind:
    if cs[i] in check:
        merge_ind.append(cs.index(cs[i]))
        combine.append(str(cs.index(cs[i]))+str(i))
for i in range(len(combine)):
    combine.append("s"+combine[i])
table=make_table(cs)
sym_list = nt_list + t_list
for i in ind:
    val = ind.index(i)
    for j in table[i]:
        if j not in table[int(merge_ind[val])]:
            table[int(merge_ind[val])][j] = table[i][j]
    table.pop(i)
for i in range(len(ind)):
    s_list = []
    s = "s" + str(ind[i])
    s_list.append(s)
    ind.append(set(s_list))
for i in range(len(merge_ind)):
    s_list = []
    s = "s" + str(merge_ind[i])
    s_list.append(s)
    merge_ind.append(set(s_list))
for i in range(0,int(len(merge_ind)/2)):
    merge_ind[i] = str(merge_ind[i])
    ind[i] = str(ind[i])
```

```
for i in table:
    for j in table[i]:
        if (table[i][j] in ind):
            ind1 = ind.index(table[i][j])
            table[i][j] = combine[ind1]
        elif (table[i][j] in merge_ind):
            ind1 = merge_ind.index(table[i][j])
            table[i][j] = combine[ind1]
for i in items:
    if i in merge_ind:
        indexof = merge_ind.index(i)
        c = combine[indexof]
        j = ind[indexof]
        j_ind = items.index(j)
        items.pop(j_ind)
        item_index = items.index(i)
        items.pop(item_index)
        items.insert(item_index,c)
print()
print("*******----STRING-----**********")
print()
lookahead = []
ctr = 0
for s in check:
    string = []
    st=[]
    if items[ctr] in combine:
        com_ind = combine.index(items[ctr])
        for j in cs[int(ind[com_ind])].closure:
            st.append(j.lookahead)
        for i in range(len(s)):
            string_i=[]
            for k in s[i].lookahead:
                string_i.append(k)
            string_i.append(st[i][0])
            string.append(string_i)
        lookahead.append(string)
    else:
        for i in range(len(s)):
            string_i=[]
            for k in s[i].lookahead:
                string_i.append(k)
            string.append(string_i)
        lookahead.append(string)
    ctr+=1
ctr = 0
for s in check:
    print("Item {}:".format(items[ctr]))
    string = ""
    for i in range(len(s)):
        string += s[i]
        string += " "
        string += str(lookahead[ctr][i])
        string += "\n"
    print(string)
    if len(items[ctr]) == 2:
        for j in table[int(items[ctr][0])]:
            if isinstance(table[int(items[ctr][0])][j],set):
                pass
            elif table[int(items[ctr][0])][j][0] == "s":
                print(j,"->",table[int(items[ctr][0])][j][1:])
            else:
                print(j,"->",table[int(items[ctr][0])][j])
    else:
        for j in table[int(items[ctr])]:
            if isinstance(table[int(items[ctr])][j],set):
                pass
            elif table[int(items[ctr][0])][j][0] == "s":
                print(j,"->",table[int(items[ctr])][j][1:])
            else:
                print(j,"->",table[int(items[ctr])][j])
    print()
    ctr+=1
dis_arr = []
print("*******----PARSING TABLE-----**********")
print('_____________________________________________________________________')
print("LALR(1) TABLE")
sym_list = nt_list + t_list
sr, rr=0, 0
print("\t      GOTO \t\t ACTION")
print('____________________________________________________________________')
print('\t|  ','\t|  '.join(sym_list),'\t\t|')
print('_____________________________________________________________________')
for i, j in table.items():
    inti = str(i)
    if inti in merge_ind:
        inti = combine[merge_ind.index(inti)]
    print(inti, "\t|  ", '\t|  '.join(list(j.get(sym,' ') if type(j.get(sym))in (str , None) else
next(iter(j.get(sym,' ')))  for sym in sym_list)),'\t\t|')
    s, r=0, 0
    dis_arr.append(inti)
    for p in j.values():
        if p!='accept' and len(p)>1:
            p=list(p)
            if('r' in p[0]): r+=1
            else: s+=1
            if('r' in p[1]): r+=1
            else: s+=1
    if r>0 and s>0: sr+=1
    elif r>0: rr+=1
print('______________________________________________________________________')
print()
dfa={}
counter = 0
for i,j in table.items():
    od={}
    for k,l in j.items():
        if isinstance(l,set):
            od[k]=''.join(l)
        elif l.isdigit():
            od[k]=int(l)
        else:
            od[k]=l
    dfa[dis_arr[counter]]=od
    counter+=1
print("*******----STRING PARSING-----**********")
print()
string=input('Enter string to parse: ')
string+='$'
stack=['0']
pointer=0
try:
    while True:
        lookahead=string[pointer]
        if dfa[stack[-1]][lookahead][0] =='s':
            act = dfa[stack[-1]][lookahead][1:]
            stack.append(lookahead)
            stack.append(act)
            print(stack)
            pointer+=1
        elif dfa[stack[-1]][lookahead][0] =='r':
```

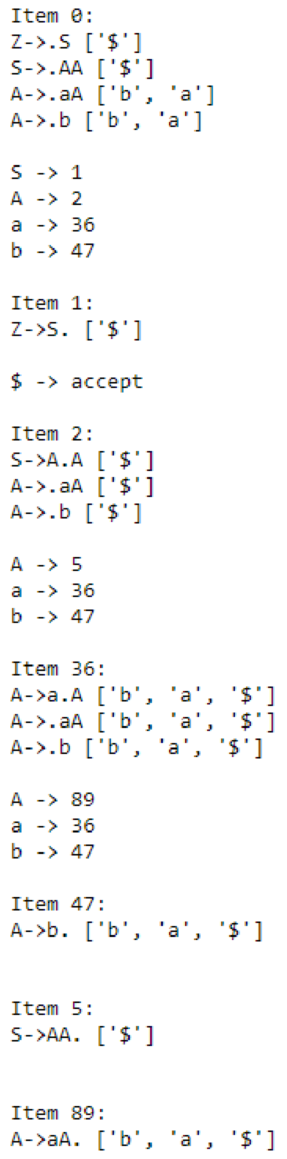
```
        r_no=int(dfa[stack[-1]][lookahead][1])
        to_pop=pro[r_no-1][3:]
        for i in range(2*len(to_pop)):
          stack.pop()
        stack.append(pro[r_no-1][0])
        stack.append(str(dfa[stack[-2]][pro[r_no-1][0]]))
        print(stack)

      elif dfa[stack[-1]][lookahead] =='accept':
        print('Succesfull parsing')
        break
  except:
    print('Unsuccesfull parsing')
```



```
Enter the grammar productions (enter 'end' or return to stop)
#(Format: "A->Y1Y2..Yn" {Yi - single char} OR "A->" {epsilon})
S->AA
A->aA
A->b
end
S
        First:   {'b', 'a'}
        Follow:  {'$'}

A
        First:   {'b', 'a'}
        Follow:  {'b', '$', 'a'}
```



```
*******----PARSING TABLE-----**********
_____________________________________________________________
LALR(1) TABLE
             GOTO              ACTION
____________________________________________________________
       |  S   |  A    |  a    |  b    |  $            |
_____________________________________________________________
0      |  1   |  2    |  s36  |  s47  |               |
1      |      |       |       |       |  accept       |      |
2      |      |  5    |  s36  |  s47  |               |
36     |      |  89   |  s36  |  s47  |               |
47     |      |       |  r3   |  r3   |  r3           |
5      |      |       |       |       |  r1           |
89     |      |       |  r2   |  r2   |  r2           |
_____________________________________________________________

*******----STRING PARSING-----**********

Enter string to parse: bb
['0', 'b', '47']
['0', 'A', '2']
['0', 'A', '2', 'b', '47']
['0', 'A', '2', 'A', '5']
['0', 'S', '1']
Succesfull parsing
```



```
Item 0:
Z->.S ['$']
S->.AA ['$']
A->.aA ['b', 'a']
A->.b ['b', 'a']

S -> 1
A -> 2
a -> 36
b -> 47

Item 1:
Z->S. ['$']

$ -> accept

Item 2:
S->A.A ['$']
A->.aA ['$']
A->.b ['$']

A -> 5
a -> 36
b -> 47

Item 36:
A->a.A ['b', 'a', '$']
A->.aA ['b', 'a', '$']
A->.b ['b', 'a', '$']

A -> 89
a -> 36
b -> 47

Item 47:
A->b. ['b', 'a', '$']


Item 5:
S->AA. ['$']


Item 89:
A->aA. ['b', 'a', '$']
```