System Software

Nehal Jhajharia (U20CS093)
Lab Assignment 7

Q)

Write a program to construct LR (1) parse table for the following grammar and check whether the given input can be accepted or not.

Grammar:

S -> AaAb | BbBa A -> ε

B -> ε

clr.py

```
from collections import deque
from collections import OrderedDict
from pprint import pprint
import firstfollow
from firstfollow import production_list, nt_list as ntl, t_list as tl
nt_list, t_list=[], []

'''
S->AaAb
S->BbBa
A->
B->
end
'''

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):

    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

def main():

    global production_list, ntl, nt_list, tl, t_list

    firstfollow.main()

    print("\tFIRST AND FOLLOW OF NON-TERMINALS")
    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()) + ['$']

    print(nt_list)
    print(t_list)

    j=calc_states()

    ctr=0
    for s in j:
        print("Item{}:".format(ctr))
        for i in s:
            print("\t", i)
        ctr+=1

    table=make_table(j)
    print('_____________________________________________________________________')
    print("\n\tCLR(1) TABLE\n")
    sym_list = nt_list + t_list
    sr, rr=0, 0
    print('_____________________________________________________________________')
    print('\t|  ','\t|  '.join(sym_list),'\t\t|')
    print('_____________________________________________________________________')
    for i, j in table.items():

        print(i, "\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

        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("\n", sr, "s/r conflicts |", rr, "r/r conflicts")
    print('_____________________________________________________________________')
    print("Enter the string to be parsed")
    Input=input()+'$'
    try:
        stack=['0']
        a=list(table.items())
        '''print(a[int(stack[-1])][1][Input[0]])
        b=list(a[int(stack[-1])][1][Input[0]])
        print(b[0][0])
        print(a[0][1]["S"])'''
        print("productions\t:",production_list)
        print('stack',"\t \t\t \t",'Input')
        print(*stack,"\t \t\t \t",*Input,sep="")
        while(len(Input)!=0):
            b=list(a[int(stack[-1])][1][Input[0]])
            if(b[0][0]=="s" ):
                #s=Input[0]+b[0][1:]
                stack.append(Input[0])
                stack.append(b[0][1:])
                Input=Input[1:]
                print(*stack,"\t \t\t \t",*Input,sep="")
            elif(b[0][0]=="r" ):
                s=int(b[0][1:])
                #print(len(production_list),s)
                l=len(production_list[s])-3
                #print(l)
                prod=production_list[s]
                l*=2
                l=len(stack)-l
                stack=stack[:l]
                s=a[int(stack[-1])][1][prod[0]]
                #print(s,b)
                stack+=list(prod[0])
                stack.append(s)
                print(*stack,"\t \t\t \t",*Input,sep="")
            elif(b[0][0]=="a"):
                print("\n\tString Accepted\n")
                break
    except:
        print('\n\tString INCORRECT for given Grammar!\n')
    return

if __name__=="__main__":
    main()
```

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 X is a terminal then first(X) = X
   if symbol in t_list:
       return set(symbol)

   for prod in production_list:
       head, body=prod.split('->')

       if head!=symbol: continue

# if X -> is a production, then first(X) = epsilon
```

```
       if body=='':
           nt_list[symbol].add_first(chr(1013))
           continue



       for i, Y in enumerate(body):
# for X -> Y1 Y2 ... Yn, first(X) = non-epsilon symbols in first(Y1)
# if first(Y1) contains epsilon,
#   first(X) = non-epsilon symbols in first(Y2)
#   if first(Y2) contains epsilon
#   ...
           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
# for i=1 to n, if Yi contains epsilon, then first(X)=epsilon
           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 A is the start symbol, follow (A) = $
   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

# for A -> aBb, follow(B) = non-epsilon symbols in first(b)
           if i != len(body)-1:
               nt_list[symbol].add_follow(get_first(body[i+1]) - set(chr(1013)))

# if A -> aBb where first(b) contains epsilon, or A -> aB then follow(B) = follow (A)
           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

   '''if pl!=None:
       for i, prod in enumerate(pl):
           if prod.lower() in ['end', '']:
               del pl[i:]
               break
           head, body=prod.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 finditer(t_regex, body):
               s=i.group()
               if s not in t_list.keys(): t_list[s]=Terminal(s)
           #for all non-terminals in the body of the production
           for i in finditer(nt_regex, body):
               s=i.group()
               if s not in nt_list.keys(): nt_list[s]=NonTerminal(s)'''

   return pl
# ------------------------------------------------------------------

if __name__=='__main__':
    main()
```

```
jhajharia@Nehals-MacBook-Air Asmt7 % python3 clr.py
Enter the grammar productions (enter 'end' or return to stop)
#(Format: "A->Y1Y2..Yn" {Yi - single char} OR "A->" {epsilon})
S->AaAb
S->BbBa
A->
B->
end
        FIRST AND FOLLOW OF NON-TERMINALS
S
        First:   {'a', 'b'}
        Follow:  {'$'}

A
        First:   {'ϵ'}
        Follow:  {'a', 'b'}

B
        First:   {'ϵ'}
        Follow:  {'a', 'b'}

['S', 'A', 'B']
['a', 'b', '$']
Item0:
         Z->.S, $
         S->.AaAb, $
         S->.BbBa, $
         A->., a
         B->., b
Item1:
         Z->S., $
Item2:
```

S->A.aAb, $

Item3:

S->B.bBa, $

Item4:

S->Aa.Ab, $
A->., b

Item5:

S->Bb.Ba, $
B->., a

Item6:

S->AaA.b, $

Item7:

S->BbB.a, $

Item8:

S->AaAb., $

Item9:

S->BbBa., $

CLR(1) TABLE

	S	A	B	a	b	$
0	1	2	3	r3	r4	
1						accept
2				s4		
3					s5	
4		6			r3	
5			7	r4		
6					s8	
7				s9		
8						r1
9						r2

```
 0 s/r conflicts | 0 r/r conflicts

Enter the string to be parsed
ab
productions    : ['Z->S', 'S->AaAb', 'S->BbBa', 'A->', 'B->']
stack                 Input
0                  ab$
0A2                  ab$
0A2a4                 b$
0A2a4A6                 b$
0A2a4A6b8                     $
0S1                  $

    String Accepted

jhajharia@Nehals-MacBook-Air Asmt7 %
```