**Lab Assignment-2**

**Compiler Construction (UCS802)**

****

**Submitted by: Lakshay Bansal**

**Roll no.: 102103090**

**Group: 4CO3**

**Department of Computer Science and Engineering**

**Thapar Institute of Engineering and Technology Patiala, Punjab – 147004**

**Aug –Dec 2024**

**Design a SLR parser for any grammar (Generic)**

**For Example: E→E+T|T**

**T→T\*F|F**

**F→(E)|i**

**Code:** (in Python)

grammar=[]

new\_grammar=[]

terminals=[]

non\_terminals=[]

I\_n={}

shift\_list=[]

reduction\_list=[]

action\_list=[]

rule\_dict={}

follow\_dict={}

SR=[]

RR=[]

def Conflict():

global SR, RR, shift\_list, reduction\_list

conflict=False

for S in shift\_list:

for R in reduction\_list:

if S[:2]==R[:2]:

SR.append([S, R])

conflict=True

for R1 in reduction\_list:

for R2 in reduction\_list:

if R1==R2:

continue

if R1[:2]==R2[:2]:

RR.append(R1)

conflict=True

return conflict

def read\_grammar():

global grammar, terminals, non\_terminals, rule\_dict

print("Enter grammar rules(press Enter)")

print("Please enter each rule separately, using 'i' instead of 'id' and avoiding the use of the '|' symbol.")

while True:

rule=input("Enter rule: ").strip()

if not rule:

break

grammar.append(rule)

for each\_grammar in grammar:

if each\_grammar[0] not in non\_terminals:

non\_terminals.append(each\_grammar[0])

for each\_grammar in grammar:

for token in each\_grammar.replace(" ", "").replace("->", ""):

if token not in non\_terminals and token not in terminals:

terminals.append(token)

for i, rule in enumerate(grammar, 1):

rule\_dict[i]=rule

def augmented\_grammar():

global grammar, new\_grammar

read\_grammar()

if "'" not in grammar[0]:

grammar.insert(0, grammar[0][0]+"'"+"->"+grammar[0][0])

new\_grammar=[]

for each\_grammar in grammar:

idx=each\_grammar.index(">")

each\_grammar=each\_grammar[:idx+1]+"."+each\_grammar[idx+1:]

new\_grammar.append(each\_grammar)

def compute\_I0():

global new\_grammar, non\_terminals, I\_n

augmented\_grammar()

grammar2add=[new\_grammar[0]]

i=0

for each in grammar2add:

current\_pos=each.index(".")

current\_variable=each[current\_pos + 1]

if current\_variable in non\_terminals:

for each\_grammar in new\_grammar:

if each\_grammar[0]==current\_variable and each\_grammar not in grammar2add:

grammar2add.append(each\_grammar)

I\_n[i]=grammar2add

def GOTO():

global grammar, non\_terminals, terminals, I\_n, shift\_list

compute\_I0()

variables=non\_terminals+terminals

i, current\_state=0, 0

done=False

while not done:

for each\_variable in variables:

grammar2add=[]

try:

for each\_rule in I\_n[current\_state]:

if each\_rule[-1]==".":

continue

dot\_idx=each\_rule.index(".")

if each\_rule[dot\_idx + 1]==each\_variable:

rule=each\_rule.replace(".", "")

rule=rule[:dot\_idx + 1] + "."+rule[dot\_idx + 1:]

grammar2add.append(rule)

for rule in grammar2add:

dot\_idx=rule.index(".")

if rule[-1]!=".":

current\_variable=rule[dot\_idx + 1]

if current\_variable in non\_terminals:

for each\_grammar in new\_grammar:

if each\_grammar[0]==current\_variable and each\_grammar not in grammar2add:

grammar2add.append(each\_grammar)

except:

done=True

break

if grammar2add:

if grammar2add not in I\_n.values():

i+=1

I\_n[i]=grammar2add

for k, v in I\_n.items():

if grammar2add==v:

idx=k

shift\_list.append([current\_state, each\_variable, idx])

current\_state+=1

def follow(var):

global rule\_dict, follow\_dict, terminals

value=[]

if var==rule\_dict[1][0]:

value.append("$")

for rule in rule\_dict.values():

lhs, rhs=rule.split("->")

if var==rule[-1]:

for each in follow(rule[0]):

if each not in value:

value.append(each)

if var in rhs:

idx=rhs.index(var)

try:

if rhs[idx+1] in non\_terminals and rhs[idx+1]!=var:

for each in follow(rhs[idx+1]):

value.append(each)

else:

value.append(rhs[idx+1])

except:

pass

return value

def reduction():

global I\_n, rule\_dict, reduction\_list

reduction\_list.append([1, "$", "Accept"])

for item in I\_n.items():

try:

for each\_production in item[1]:

lhs, rhs=each\_production.split(".")

for rule in rule\_dict.items():

if lhs==rule[1]:

f=follow(lhs[0])

for each\_var in f:

reduction\_list.append([item[0], each\_var, "R"+str(rule[0])])

except:

pass

def test(string):

global action\_list, shift\_list, reduction\_list

done=False

stack=[0]

print("\nSTACK\t\tSTRING\t\tACTION")

while not done:

Reduce=False

Shift=False

for r in reduction\_list:

if r[0]==int(stack[-1]) and r[1]==string[0]:

Reduce=True

print("".join(str(p) for p in stack), "\t\t", string, "\t\t", "Reduce", r[2])

if r[2]=='Accept':

return 1

var=rule\_dict[int(r[2][1])]

lhs, rhs=var.split("->")

for \_ in rhs:

stack.pop()

stack.pop()

stack.append(lhs)

for a in action\_list:

if a[0]==int(stack[-2]) and a[1]==stack[-1]:

stack.append(str(a[2]))

break

for g in shift\_list:

if g[0]==int(stack[-1]) and g[1]==string[0]:

Shift=True

print("".join(str(p) for p in stack), "\t\t", string, "\t\t", "Shift", "S" + str(g[2]))

stack.append(string[0])

stack.append(str(g[2]))

string=string[1:]

if not Reduce and not Shift:

print("".join(str(p) for p in stack), string)

return 0

def main():

global I\_n, shift\_list, reduction\_list, action\_list, SR, RR

GOTO()

reduction()

print("\n\*AUGMENTED RULES\*")

for item in new\_grammar:

print(item.replace(".", ""))

print("\n\*STATES\*")

for item in I\_n.items():

print(item)

print("\n\*GOTO OPERATIONS\*")

for item in shift\_list:

print(item)

print("\n\*REDUCTION\*")

for item in reduction\_list:

print(item)

if Conflict():

if SR:

print("SR conflict")

for item in SR:

print(item)

print()

if RR:

print("RR conflict")

for item in RR:

print(item)

print()

exit(0)

else:

print("\nNO CONFLICT\n")

print("Terminals:", terminals)

print("NonTerminals:", non\_terminals)

print()

action\_list.extend(shift\_list)

action\_list.extend(reduction\_list)

string=input("\nEnter String to Parse (end with $): ")

if not string.endswith("$"):

string+="$"

print("\nTest String:", string)

result=test(string)

if result==1:

print("\n\*ACCEPTED\*")

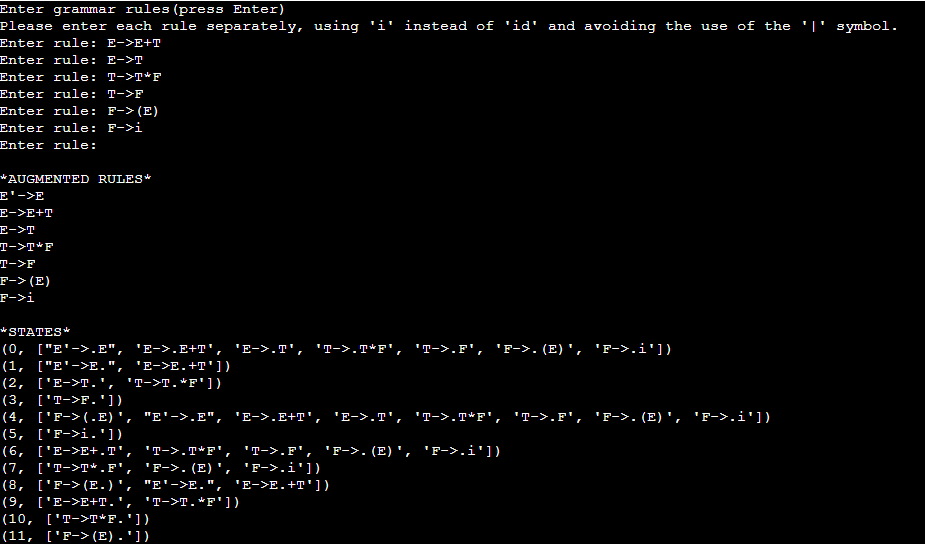
elif result==0:

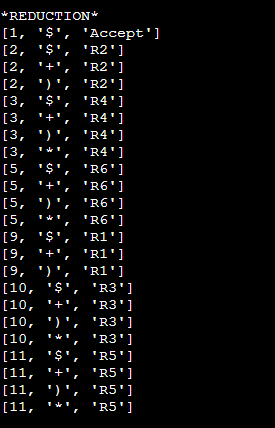
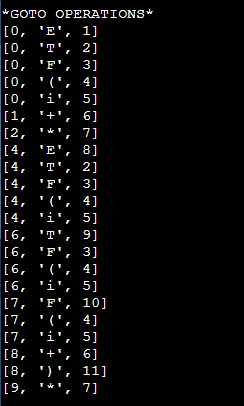
print("\n\*NOT ACCEPTED\*")

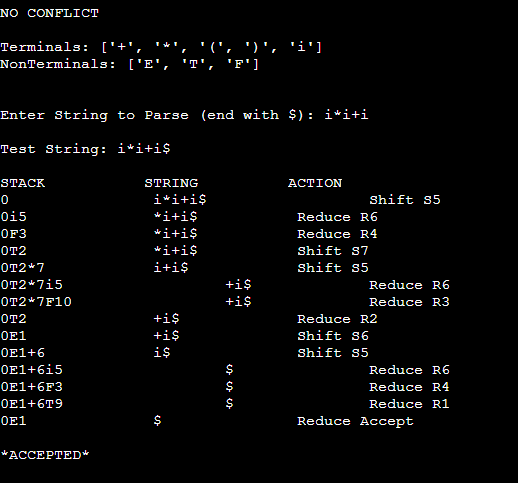
if \_\_name\_\_=='\_\_main\_\_':

main()

**Output: (Snapshots)**

****

****

****