Ishita Hardasmalani

C14-2103058

EXPERIMENT NO.3

Aim : Write a program to implement the FIRST & FOLLOW set for the given grammar

Code:

```
def computeFirst(rule, rules, nonterm_userdef, term_userdef, diction, firsts):
  if len(rule) != 0 and (rule is not None):
    if rule[0] in term_userdef:
      return [rule[0]]
    elif rule[0] == '#':
      return ['#']
    if len(rule) != 0:
      if rule[0] in list(diction.keys()):
        fres = []
        rhs_rules = diction[rule[0]]
        for itr in rhs_rules:
          indivRes = computeFirst(itr, rules, nonterm_userdef, term_userdef, diction, firsts)
          if type(indivRes) is list:
            fres.extend(indivRes)
          else:
            fres.append(indivRes)
        if '#' not in fres:
          return list(set(fres))
        else:
          fres.remove('#')
```

```
                if len(rule) > 1:
                    ansNew = computeFirst(rule[1:], rules, nonterm_userdef, term_userdef, diction, firsts)
                    if ansNew is not None:
                        if type(ansNew) is list:
                            return list(set(fres).union(set(ansNew)))
                        else:
                            return list(set(fres).union({ansNew}))
                fres.append('#')
                return list(set(fres))
def computeFollow(nt, start_symbol, rules, nonterm_userdef, term_userdef, diction, firsts, follows):
    solset = set()
    if nt == start_symbol:
        solset.add('$')
    for curNT in diction:
        rhs = diction[curNT]
        for subrule in rhs:
            if nt in subrule:
                index_nt = subrule.index(nt)
                remaining_subrule = subrule[index_nt + 1:]
                if len(remaining_subrule) != 0:
                    res = computeFirst(remaining_subrule, rules, nonterm_userdef, term_userdef, diction, firsts)
                    if '#' in res:
                        res.remove('#')
                    ansNew = computeFollow(curNT, start_symbol, rules, nonterm_userdef, term_userdef, diction, firsts, follows)
                    if ansNew is not None:
                        solset.update(res)
                        solset.update(ansNew)
                    else:
                        solset.update(res)
                else:
                    res = computeFirst(remaining_subrule, rules, nonterm_userdef, term_userdef, diction, firsts)
                    solset.update(res)
                    if len(remaining_subrule) == 0 or '#' in computeFirst(remaining_subrule, rules, nonterm_userdef, term_userdef, diction, firsts):
                        if nt != curNT:
                            ansNew = computeFollow(curNT, start_symbol, rules, nonterm_userdef, term_userdef, diction, firsts, follows)
                            if ansNew is not None:
                                solset.update(ansNew)
    return list(solset)


# Example usage
rules = ["S->AS|C", "A->a|b|Bc", "B->p|a", "C->c"]
nonterm_userdef = ['S', 'A', 'B', 'C']
term_userdef = ['a', 'c', 'b', 'p']
diction = {}
firsts = {}
follows = {}


# computeFirst for each rule
```

```
for rule in rules:
    k = rule.split("->")
    k[0] = k[0].strip()
    k[1] = k[1].strip()
    rhs = k[1]
    multirhs = rhs.split('|')
    for i in range(len(multirhs)):
        multirhs[i] = multirhs[i].strip()
        multirhs[i] = multirhs[i].split()
    diction[k[0]] = multirhs

# computeFirst for each non-terminal
for y in list(diction.keys()):
    t = set()
    for sub in diction.get(y):
        res = computeFirst(sub, rules, nonterm_userdef, term_userdef, diction,
firsts)
        if res is not None:
            if type(res) is list:
                for u in res:
                    t.add(u)
            else:
                t.add(res)
    firsts[y] = t

# computeFollow for each non-terminal
for NT in diction:
    solset = set()
    sol = computeFollow(NT, list(diction.keys())[0], rules, nonterm_userdef,
term_userdef, diction, firsts, follows)
    if sol is not None:
        for g in sol:
            solset.add(g)
    follows[NT] = solset

# Print the results
print("\nCalculated firsts:")
key_list = list(firsts.keys())
index = 0
for g in firsts:
    print(f"first({key_list[index]})=>{firsts.get(g)}")
    index += 1
print("\nCalculated follows:")
key_list = list(follows.keys())
index = 0
for g in follows:
    print(f"follow({key_list[index]})=>{follows[g]}")
    index += 1
```

Output:

```
Calculated firsts: first(S) => {'c', 'p', 'b', 'a'} first(A) => {'p', 'a', 'b'} first(B) => {'p', 'a'} first(C) => {'c'}
Calculated follows: follow(S) => {'$'} follow(A) => {'c', 'p', 'b', 'a'} follow(B) => {'c'} follow(C) => {'$'}
```