**Compiler Design Lab**

**Practical-3**

**Name: Yash Patni**

**Roll: 71**

**Batch: A4**

**Aim:** (A) Write a program to find FIRST for any grammar. All the following rules of FIRST must be implemented.

(B) Calculate Follow for the given grammar and Construct the LL (1) parsing table using the FIRST and FOLLOW.

**Code:**

import sys

from tabulate import tabulate

terminals = ['a', 'b', 'c', 'p']

non\_terminals = ['S', 'A', 'B', 'C']

starting\_symbol = 'S'

productions = ["S->ABC/C", "A->a/bB/#", "B->p/#", "C->c"]

productions\_dict = {nT: [] for nT in non\_terminals}

for production in productions:

    nonterm\_to\_prod = production.split("->")

    alternatives = nonterm\_to\_prod[1].split("/")

    for alternative in alternatives:

        productions\_dict[nonterm\_to\_prod[0]].append(alternative)

LL1\_table = {}

conflicts = False

def first(string):

    first\_ = set()

    if string in non\_terminals:

        alternatives = productions\_dict[string]

        for alternative in alternatives:

            first\_2 = first(alternative)

            first\_ |= first\_2

    elif string in terminals:

        first\_ = {string}

    elif string == '' or string == '#':

        first\_ = {'#'}

    else:

        first\_2 = first(string[0])

        if '#' in first\_2:

            i = 1

            while '#' in first\_2:

                first\_ |= (first\_2 - {'#'})

                if string[i:] in terminals:

                    first\_ |= {string[i:]}

                    break

                elif string[i:] == '':

                    first\_ |= {'#'}

                    break

                first\_2 = first(string[i:])

                first\_ |= (first\_2 - {'#'})

                i += 1

        else:

            first\_ |= first\_2

    return first\_

def follow(nT):

    follow\_ = set()

    prods = productions\_dict.items()

    if nT == starting\_symbol:

        follow\_ |= {'$'}

    for nt, rhs in prods:

        for alt in rhs:

            for char in alt:

                if char == nT:

                    following\_str = alt[alt.index(char) + 1:]

                    if following\_str == '':

                        if nt == nT:

                            continue

                        else:

                            follow\_ |= follow(nt)

                    else:

                        follow\_2 = first(following\_str)

                        if '#' in follow\_2:

                            follow\_ |= (follow\_2 - {'#'})

                            follow\_ |= follow(nt)

                        else:

                            follow\_ |= follow\_2

    return follow\_

FIRST = {non\_terminal: set() for non\_terminal in non\_terminals}

FOLLOW = {non\_terminal: set() for non\_terminal in non\_terminals}

for non\_terminal in non\_terminals:

    FIRST[non\_terminal] |= first(non\_terminal)

FOLLOW[starting\_symbol] |= {'$'}

for non\_terminal in non\_terminals:

    FOLLOW[non\_terminal] |= follow(non\_terminal)

print("{: ^20}{: ^20}{: ^20}".format('Non Terminals', 'First', 'Follow'))

for non\_terminal in non\_terminals:

    print("{: ^20}{: ^20}{: ^20}".format(non\_terminal, str(FIRST[non\_terminal]), str(FOLLOW[non\_terminal])))

for non\_terminal, alternatives in productions\_dict.items():

    for alternative in alternatives:

        first\_set\_alt = first(alternative)

        for terminal in first\_set\_alt - {'#'}:

            if (terminal, non\_terminal) not in LL1\_table:

                LL1\_table[(terminal, non\_terminal)] = [alternative]

            else:

                LL1\_table[(terminal, non\_terminal)].append(alternative)

                if len(LL1\_table[(terminal, non\_terminal)]) > 1:

                    conflicts = True

                    print(f"Conflict at ({terminal}, {non\_terminal})")

        if '#' in first\_set\_alt or '' in first\_set\_alt:  # If epsilon is in FIRST(α)

            for terminal in FOLLOW[non\_terminal]:

                if (terminal, non\_terminal) not in LL1\_table:

                    LL1\_table[(terminal, non\_terminal)] = [alternative]

                else:

                    LL1\_table[(terminal, non\_terminal)].append(alternative)

                    if len(LL1\_table[(terminal, non\_terminal)]) > 1:

                        conflicts = True

                        print(f"Conflict at ({terminal}, {non\_terminal})")

# Displaying the LL(1) parsing table with terminals as row headers and non-terminals as column headers

table\_data = []

for terminal in terminals:

    row = [terminal] + [', '.join(LL1\_table.get((terminal, non\_terminal), [])) for non\_terminal in non\_terminals]

    table\_data.append(row)

print(tabulate(table\_data, headers=['Terminals'] + non\_terminals, tablefmt='grid'))

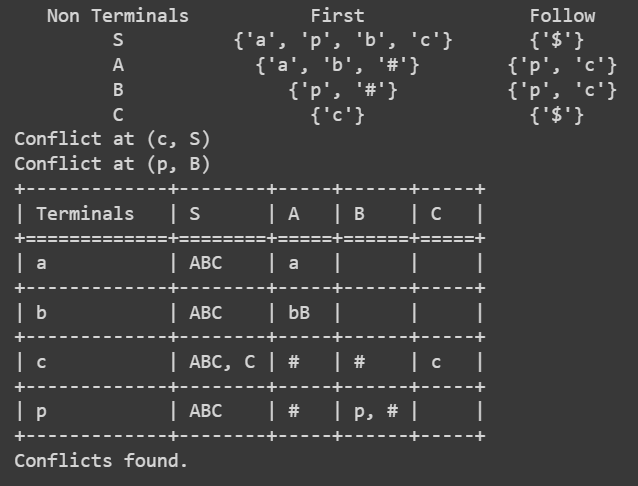
if conflicts:

    print("Conflicts found.")

else:

    print("No conflicts found.")

**Output:**

****

**Conclusion:** In this practical we successfully implemented program to calculate first and follow for the given production. Also constructed the LL(1) parsing table.