COMPILER DESIGN

EXP-7 Computation of Leading and Trailing

Tamojit Sarkar **Date:31/03/2021**
RA1811027010034
CSE-BD Sec-I2

Aim: To design a code to compute leading and trailing sets of the given grammar

Language Used : Python

Procedure:

1. Epsilon is represented by ‘e’
2. Productions are of the form A=B, where ‘A’ is a single Non-Terminal and ‘B’ can beany combination of Terminals and Non- Terminals.
3. To compute Leading(A) where A is a non terminal apply the following algorithm
   LEADING(A)
   {
   1. ‘a’ is in Leading(A) if A ->γaδ where γ is ε or any Non-Terminal
   2. If’ ‘a’ is in Leading(B) and A ->Bα, then a in Leading(A)
   }
   Step 1 of algorithm , indicates how to add the first terminal occurring in the RHS of every production directly.
   Step 2 of the algorithm indicates to add the first terminal, through another non-terminal B to be included indirectly to the LEADING() of every non-terminal.
4. To compute Trailing(A) where A is a non terminal apply the following algorithm
   Trailing(A)
   {
   1. a is in Trailing(A) if A ? γaδ where δ is ε or any Non-Terminal
   2. If a is in Trailing(B) and A ? αB, then a in Trailing(A)
   }
   Algorithm Trailing is similar to algorithm Leading and the only difference being, the symbol is looked from right to left as against left to right in algorithm Leading.
   Step 1 of the algorithm Trailing, indicates looking for the first terminal occurring in the RHS of a production from right side and thus adds the direct first symbol.
   The second step looks for adding the indirect first symbol from the right of the RHS of the production.

Code:

```
a = ["E=E+T",
     "E=T",
     "T=T*F",
     "T=F",
     "F=(E)",
     "F=i"]

rules = {}
terms = []
for i in a:
    temp = i.split("=")
    terms.append(temp[0])
    try:
        rules[temp[0]] += [temp[1]]
    except:
        rules[temp[0]] = [temp[1]]

terms = list(set(terms))
print(rules,terms)
def leading(gram, rules, term, start):
    s = []
    if gram[0] not in terms:
        return gram[0]
    elif len(gram) == 1:
        return [0]
    elif gram[1] not in terms and gram[-1] is not start:
        for i in rules[gram[-1]]:
            s+= leading(i, rules, gram[-1], start)
            s+= [gram[1]]
        return s
def trailing(gram, rules, term, start):
    s = []
    if gram[-1] not in terms:
        return gram[-1]
    elif len(gram) == 1:
        return [0]
    elif gram[-2] not in terms and gram[-1] is not start:

        for i in rules[gram[-1]]:
            s+= trailing(i, rules, gram[-1], start)
            s+= [gram[-2]]
        return s
leads = {}
trails = {}
for i in terms:
    s = [0]
    for j in rules[i]:
        s+=leading(j,rules,i,i)
    s = set(s)
    s.remove(0)
    leads[i] = s
    s = [0]
    for j in rules[i]:
        s+=trailing(j,rules,i,i)
    s = set(s)
    s.remove(0)
    trails[i] = s

for i in terms:
    print("LEADING("+i+"):",leads[i])
for i in terms:
    print("TRAILING("+i+"):",trails[i])
```

Output:

```
{'E': ['E+T', 'T'], 'T': ['T*F', 'F'], 'F': ['(E)', 'i']} ['F', 'E', 'T']
LEADING(F): {'i', '('}
LEADING(E): {'(', 'i', '*', '+'}
LEADING(T): {'*', 'i', '('}
TRAILING(F): {')', 'i'}
TRAILING(E): {'i', '*', ')', '+'}
TRAILING(T): {'*', ')', 'i'}
```

Result: Computed Leading and Trailing of given grammar successfully.