LAB ASSIGNMENT-2

Submitted for

COMPILER CONSTRUCTION (UCS802)

Submitted by

Aryan Shanker

Saxena

102103613

4 COE 22

Submitted to

Mr. Rajesh

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology, Patiala

QUESTION:

Design a SLR parser for the grammar given below:

E→E+T/T

T→T*F/F

F→(E)/id

PYTHON CODE:

```
from collections import deque
from typing import List, Dict, Tuple, Set

class Action:
    def __init__(self, type: str, state: int):
        self.type = type
        self.state = state

# Creating ACTION and GOTO tables
action_table: Dict[int, Dict[str, Action]] = {
    0: {"id": Action("shift", 5), "(": Action("shift",
4)},
    1: {"+": Action("shift", 6), "$": Action("accept",
0)},
    2: {"+": Action("reduce", 2), "*": Action("shift",
7), ")": Action("reduce", 2), "$": Action("reduce",
2)},
    3: {"+": Action("reduce", 4), "*":
Action("reduce", 4), ")": Action("reduce", 4), "$":
Action("reduce", 4)},
    4: {"id": Action("shift", 5), "(": Action("shift",
4)},
    5: {"+": Action("reduce", 6), "*":
Action("reduce", 6), ")": Action("reduce", 6), "$":
Action("reduce", 6)},
```

```
    6: {"id": Action("shift", 5), "(": Action("shift",
4)},
    7: {"id": Action("shift", 5), "(": Action("shift",
4)},
    8: {"+": Action("shift", 6), ")": Action("shift",
11)},
    9: {"+": Action("reduce", 1), "*": Action("shift",
7), ")": Action("reduce", 1), "$": Action("reduce",
1)},
    10: {"+": Action("reduce", 3), "*":
Action("reduce", 3), ")": Action("reduce", 3), "$":
Action("reduce", 3)},
    11: {"+": Action("reduce", 5), "*":
Action("reduce", 5), ")": Action("reduce", 5), "$":
Action("reduce", 5)},
}

goto_table: Dict[int, Dict[str, int]] = {
    0: {"E": 1, "T": 2, "F": 3},
    4: {"E": 8, "T": 2, "F": 3},
    6: {"T": 9, "F": 3},
    7: {"F": 10},
}

productions: List[Tuple[str, int]] = [
    ("E'", 1),  # Dummy production to initialize the
parse
    ("E", 3),   # E -> E + T
    ("E", 1),   # E -> T
    ("T", 3),   # T -> T * F
    ("T", 1),   # T -> F
    ("F", 3),   # F -> ( E )
    ("F", 1)    # F -> id
]
```

```
# FIRST and FOLLOW sets for non-terminals
first_set: Dict[str, Set[str]] = {
    "E": {"id", "("},
    "T": {"id", "("},
    "F": {"id", "("},
}

follow_set: Dict[str, Set[str]] = {
    "E": {")", "+", "$"},
    "T": {"+", "*", ")", "$"},
    "F": {"*", "+", ")", "$"},
}

def print_first_and_follow_sets():
    print("FIRST Sets:")
    for non_terminal, symbols in first_set.items():
        print(f"FIRST({non_terminal}) = {{ {'
'.join(symbols)} }}")

    print("\nFOLLOW Sets:")
    for non_terminal, symbols in follow_set.items():
        print(f"FOLLOW({non_terminal}) = {{ {'
'.join(symbols)} }}")

def print_action_and_goto_tables():
    terminals = ["id", "+", "*", "(", ")", "$"]
    non_terminals = ["E", "T", "F"]

    print("\nACTION Table:")
    print(f"{'State':<8}", end="")
    for term in terminals:
        print(f"{term:<8}", end="")
    print("\n" + "-" * (8 + len(terminals) * 8))
```

```
    for state in range(12):
        print(f"{state:<8}", end="")
        for term in terminals:
            action = action_table.get(state,
{}).get(term)
            if action:
                if action.type == "shift":
                    print(f"s{action.state:<7}",
end="")
                elif action.type == "reduce":
                    print(f"r{action.state:<7}",
end="")
                elif action.type == "accept":
                    print(f"acc{'':<6}", end="")
            else:
                print(" " * 8, end="")
        print()

    print("\nGOTO Table:")
    print(f"{'State':<8}", end="")
    for non_term in non_terminals:
        print(f"{non_term:<8}", end="")
    print("\n" + "-" * (8 + len(non_terminals) * 8))

    for state in range(12):
        print(f"{state:<8}", end="")
        for non_term in non_terminals:
            next_state = goto_table.get(state,
{}).get(non_term, " ")
            print(f"{next_state:<8}", end="")
        print()

def slr_parser(tokens: List[str]):
```

```
    state_stack = deque([0])
    symbol_stack = deque()

    i = 0
    while True:
        state = state_stack[-1]
        token = tokens[i]

        if token not in action_table.get(state, {}):
            print("Status: Rejected")
            return

        action = action_table[state][token]

        if action.type == "shift":
            state_stack.append(action.state)
            symbol_stack.append(token)
            i += 1
        elif action.type == "reduce":
            production_idx = action.state
            prod_len = productions[production_idx][1]
            non_terminal =
productions[production_idx][0]

            for _ in range(prod_len):
                state_stack.pop()
                symbol_stack.pop()

            next_state = goto_table[state_stack[-
1]][non_terminal]
            state_stack.append(next_state)
            symbol_stack.append(non_terminal)
        elif action.type == "accept":
            print("Status: Accepted")
```

```
            return

if __name__ == "__main__":

    print_first_and_follow_sets()
    print_action_and_goto_tables()

    print("\nThe input string to parse: id + id * id")
    input_tokens = ["id", "+", "id", "*", "id", "$"]
    slr_parser(input_tokens)
```

OUTPUT:

Output:

```
FIRST Sets:
FIRST(E) = { ( id }
FIRST(T) = { ( id }
FIRST(F) = { ( id }

FOLLOW Sets:
FOLLOW(E) = { + $ ) }
FOLLOW(T) = { + * $ ) }
FOLLOW(F) = { + * $ ) }

ACTION Table:
State   id      +       *       (       )       $
------------------------------------------------------------
0       s5                      s4
1               s6                              acc
2               r2      s7              r2      r2
3               r4      r4              r4      r4
4       s5                      s4
5               r6      r6              r6      r6
6       s5                      s4
7       s5                      s4
8               s6                      s11
9               r1      s7              r1      r1
10              r3      r3              r3      r3
11              r5      r5              r5      r5

GOTO Table:
State   E       T       F
--------------------------------
0       1       2       3
1
2
3
4       8       2       3
5
6               9       3
7                       10
8
9
10
11
```

For id * id + id:

```
The input string to parse: id + id * id
Status: Accepted
```

For id * id + F:

```
The input string to parse: id + id * F
Status: Rejected
```