COMPILER CONSTRUCTION
ASSIGNMENT-2
UCS802

Computer Science and Engineering Department
Thapar Institute of Engineering and Technology
(Deemed to be University), Patiala – 147004

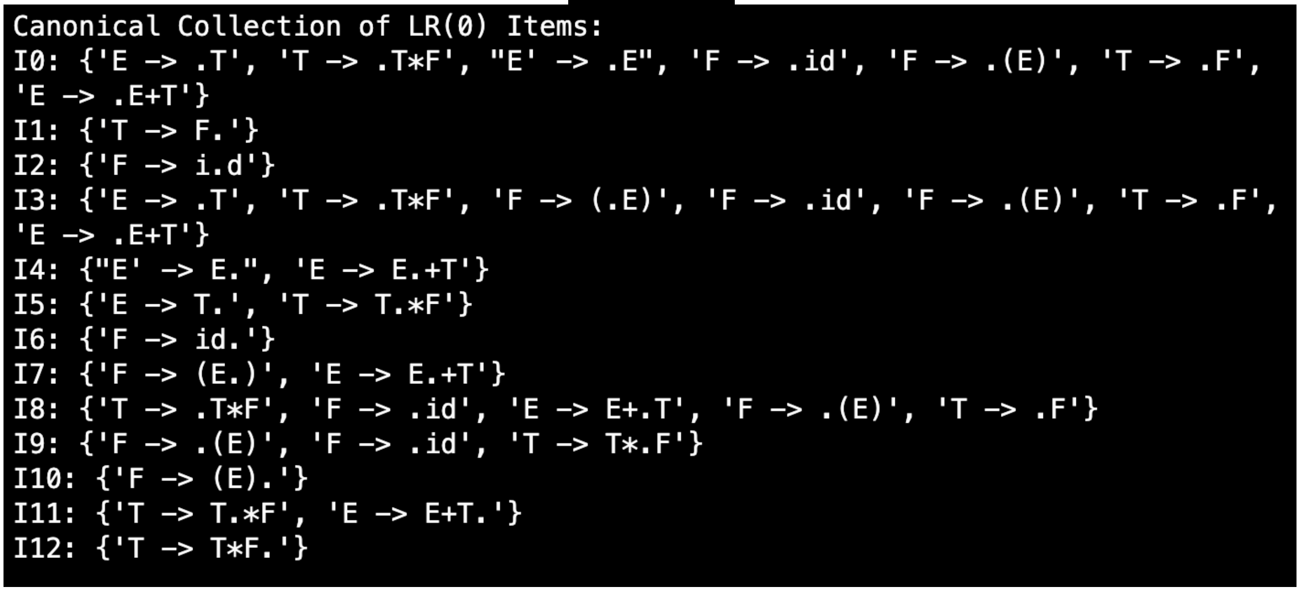
Submitted By:

Yashas Garg
4CS1
102117012

Part-1: Generating Set of Items

```
from graphviz import Digraph
from collections import defaultdict

# --- Part 1: Generate Set of Items (LR(0) Items) ---

# Define the grammar
grammar = {
    "E'": ["E"],
    "E": ["E+T", "T"],
    "T": ["T*F", "F"],
    "F": ["(E)", "id"]
}

# Closure operation for LR(0) items
def closure(items):
    closure_set = set(items)
    added = True
    while added:
        added = False
        new_items = set(closure_set)
        for item in closure_set:
            lhs, rhs = item.split(" -> ")
            pos = rhs.find(".")
            if pos < len(rhs) - 1:
                symbol = rhs[pos + 1]
                if symbol in grammar:
                    for production in grammar[symbol]:
                        new_item = f"{symbol} -> .{production}"
                        if new_item not in new_items:
                            new_items.add(new_item)
                            added = True
        closure_set = new_items
    return closure_set

# GOTO function for LR(0) items
def goto(items, symbol):
    next_items = set()
    for item in items:
        lhs, rhs = item.split(" -> ")
        pos = rhs.find(".")
        if pos < len(rhs) - 1 and rhs[pos + 1] == symbol:
            new_rhs = rhs[:pos] + symbol + "." + rhs[pos + 2:]
            next_items.add(f"{lhs} -> {new_rhs}")
    return closure(next_items)

# Generating the Canonical Collection of LR(0) Items
def canonical_collection():
    states = []
    start_item = closure(["E' -> .E"])
    states.append(start_item)
    transitions = {}
    added = True
    while added:
        added = False
        new_states = list(states)
        for i, state in enumerate(states):
            symbols = set(symbol for item in state for symbol in item if
symbol.isalpha() or symbol in "+*()")
            for symbol in symbols:
                new_state = goto(state, symbol)
                if new_state and new_state not in new_states:
                    new_states.append(new_state)
                    transitions[(i, symbol)] = len(new_states) - 1
                    added = True
                elif new_state:
                    transitions[(i, symbol)] = new_states.index(new_state)
        states = new_states
    return states, transitions
```

Output

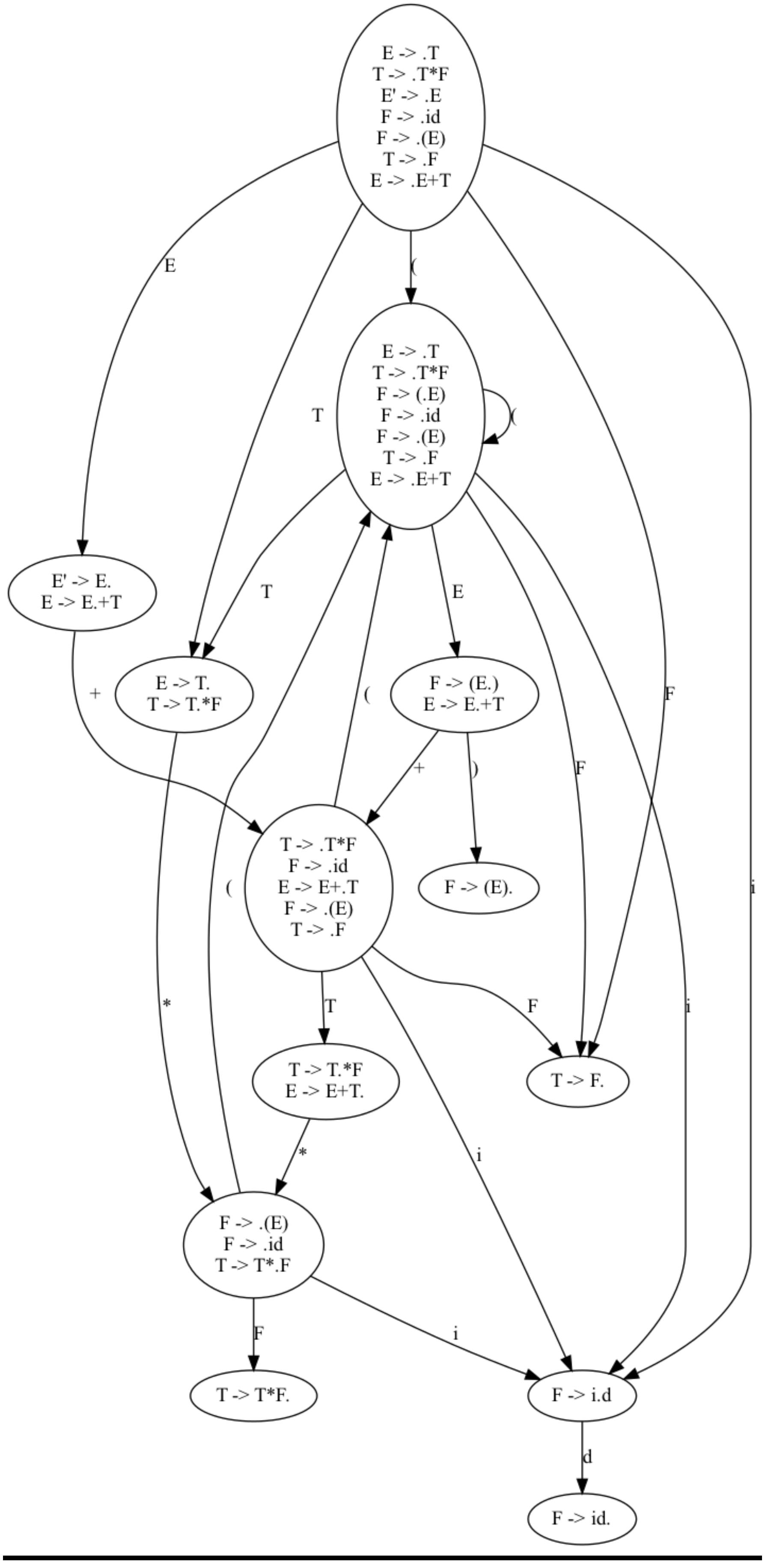


```
Canonical Collection of LR(0) Items:
I0: {'E -> .T', 'T -> .T*F', "E' -> .E", 'F -> .id', 'F -> .(E)', 'T -> .F',
'E -> .E+T'}
I1: {'T -> F.'}
I2: {'F -> i.d'}
I3: {'E -> .T', 'T -> .T*F', 'F -> (.E)', 'F -> .id', 'F -> .(E)', 'T -> .F',
'E -> .E+T'}
I4: {"E' -> E.", 'E -> E.+T'}
I5: {'E -> T.', 'T -> T.*F'}
I6: {'F -> id.'}
I7: {'F -> (E.)', 'E -> E.+T'}
I8: {'T -> .T*F', 'F -> .id', 'E -> E+.T', 'F -> .(E)', 'T -> .F'}
I9: {'F -> .(E)', 'F -> .id', 'T -> T*.F'}
I10: {'F -> (E).'}
I11: {'T -> T.*F', 'E -> E+T.'}
I12: {'T -> T*F.'}
```

Visualisation of LR(0) Items

```
# Run and visualize states
states, transitions = canonical_collection()
print("Canonical Collection of LR(0) Items:")
for i, state in enumerate(states):
    print(f"I{i}: {state}")

# Visualization of States and Transitions
dot = Digraph(comment="SLR Parser States")
for i, state in enumerate(states):
    state_label = "\n".join(state)
    dot.node(f"I{i}", label=state_label)
for (i, symbol), j in transitions.items():
    dot.edge(f"I{i}", f"I{j}", label=symbol)
dot.render("canonical_collection", format="png", view=True)
```

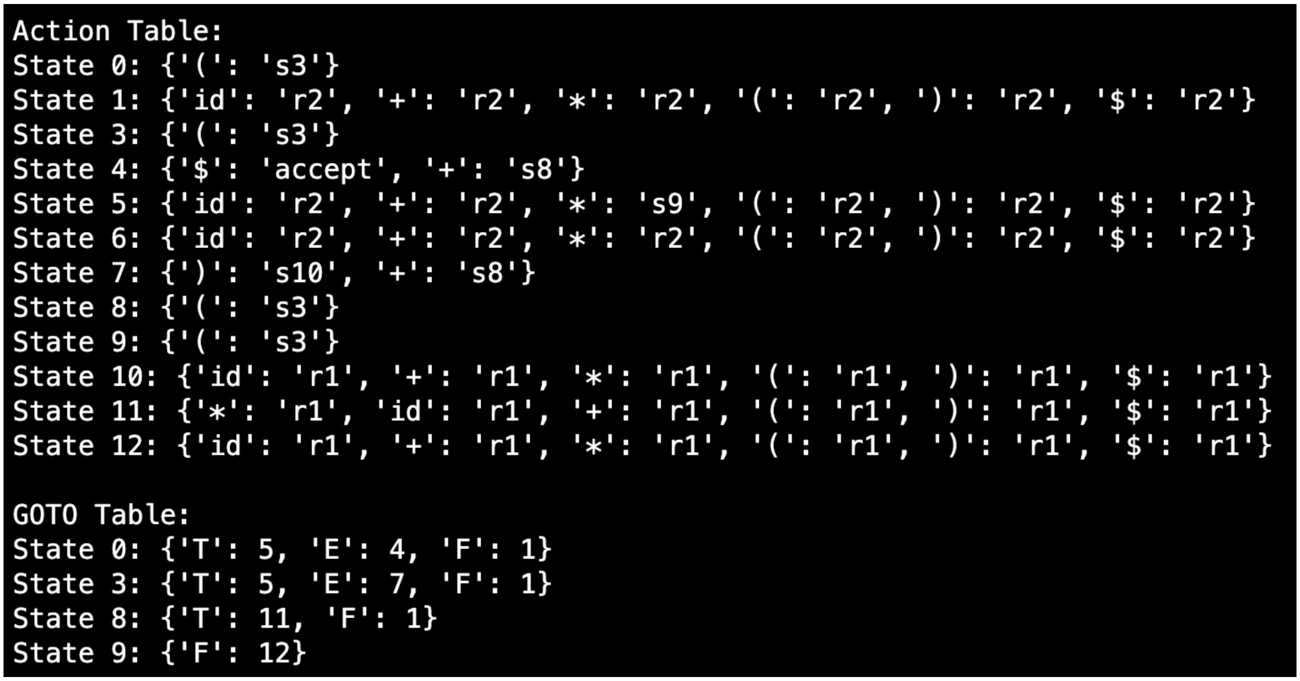
Output


E -> .T
T -> .T*F
E' -> .E
F -> .id
F -> .(E)
T -> .F
E -> .E+T
E
(
T
E -> .T
T -> .T*F
F -> (.E)
F -> .id
F -> .(E)
T -> .F
E -> .E+T
(
E' -> E.
E -> E.+T
T
E
E -> T.
T -> T.*F
+
(
F -> (E.)
E -> E.+T
F
+
)
F
T -> .T*F
F -> .id
E -> E+.T
F -> .(E)
T -> .F
(
F -> (E).
i
*
T
F
i
T -> T.*F
E -> E+T.
T -> F.
*
i
F -> .(E)
F -> .id
T -> T*.F
F
i
T -> T*F.
F -> i.d
d
F -> id.

Part-2: Generating Action & GOTO Tables

```
# --- Part 2: Constructing Action and GOTO Tables ---

# Define terminal and non-terminal symbols
terminals = ['id', '+', '*', '(', ')', '$']
non_terminals = ['E', 'T', 'F']

# Initialize Action and GOTO tables
action_table = defaultdict(lambda: defaultdict(str))
goto_table = defaultdict(lambda: defaultdict(str))

# Fill Action and GOTO tables based on parsing rules
for i, state in enumerate(states):
    for item in state:
        lhs, rhs = item.split(" -> ")
        if rhs.endswith("."):
            if lhs == "E'":
                action_table[i]['$'] = 'accept'
            else:
                prod_num = next((k for k, v in enumerate(grammar[lhs]) if v == rhs[:-
1]), None)
                for term in terminals:
                    action_table[i][term] = f"r{prod_num + 1}"
        else:
            next_symbol = rhs[rhs.find(".") + 1]
            if next_symbol in terminals:
                action_table[i][next_symbol] = f"s{transitions.get((i, next_symbol),
'')}"
            elif next_symbol in non_terminals:
                goto_table[i][next_symbol] = transitions.get((i, next_symbol), "")

print("\nAction Table:")
for state in action_table:
    print(f"State {state}: {dict(action_table[state])}")

print("\nGOTO Table:")
for state in goto_table:
    print(f"State {state}: {dict(goto_table[state])}")
```

Output

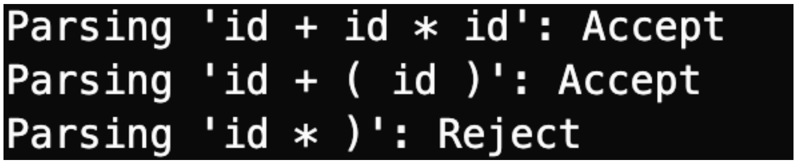


```
Action Table:
State 0: {'(': 's3'}
State 1: {'id': 'r2', '+': 'r2', '*': 'r2', '(': 'r2', ')': 'r2', '$': 'r2'}
State 3: {'(': 's3'}
State 4: {'$': 'accept', '+': 's8'}
State 5: {'id': 'r2', '+': 'r2', '*': 's9', '(': 'r2', ')': 'r2', '$': 'r2'}
State 6: {'id': 'r2', '+': 'r2', '*': 'r2', '(': 'r2', ')': 'r2', '$': 'r2'}
State 7: {')': 's10', '+': 's8'}
State 8: {'(': 's3'}
State 9: {'(': 's3'}
State 10: {'id': 'r1', '+': 'r1', '*': 'r1', '(': 'r1', ')': 'r1', '$': 'r1'}
State 11: {'*': 'r1', 'id': 'r1', '+': 'r1', '(': 'r1', ')': 'r1', '$': 'r1'}
State 12: {'id': 'r1', '+': 'r1', '*': 'r1', '(': 'r1', ')': 'r1', '$': 'r1'}

GOTO Table:
State 0: {'T': 5, 'E': 4, 'F': 1}
State 3: {'T': 5, 'E': 7, 'F': 1}
State 8: {'T': 11, 'F': 1}
State 9: {'F': 12}
```

Part-3: Parsing & Results

```
# --- Part 3: Implement the Parsing Algorithm ---

def parse(input_string):
    stack = [0]  # Start with the initial state
    tokens = input_string.split() + ['$']  # Add end symbol
    pointer = 0  # Input pointer

    while True:
        state = stack[-1]
        token = tokens[pointer]
        action = action_table[state].get(token, '')

        if action.startswith("s"):
            # Shift action
            stack.append(token)
            stack.append(int(action[1:]))
            pointer += 1
        elif action.startswith("r"):
            # Reduce action
            prod_num = int(action[1:])
            lhs, rhs = list(grammar.keys())[prod_num - 1],
grammar[list(grammar.keys())[prod_num - 1]][prod_num - 1]
            for _ in range(2 * len(rhs)):
                stack.pop()  # Pop symbols and states for reduction
            stack.append(lhs)
            stack.append(goto_table[stack[-2]][lhs])
        elif action == "accept":
            return "Accept"
        else:
            return "Reject"

# Test the parser
print("\nParsing 'id + id * id':", parse("id + id * id"))
print("Parsing 'id + ( id )':", parse("id + ( id )"))
print("Parsing 'id * )':", parse("id * )"))
```

Output



```
Parsing 'id + id * id': Accept
Parsing 'id + ( id )': Accept
Parsing 'id * )': Reject
```