

**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

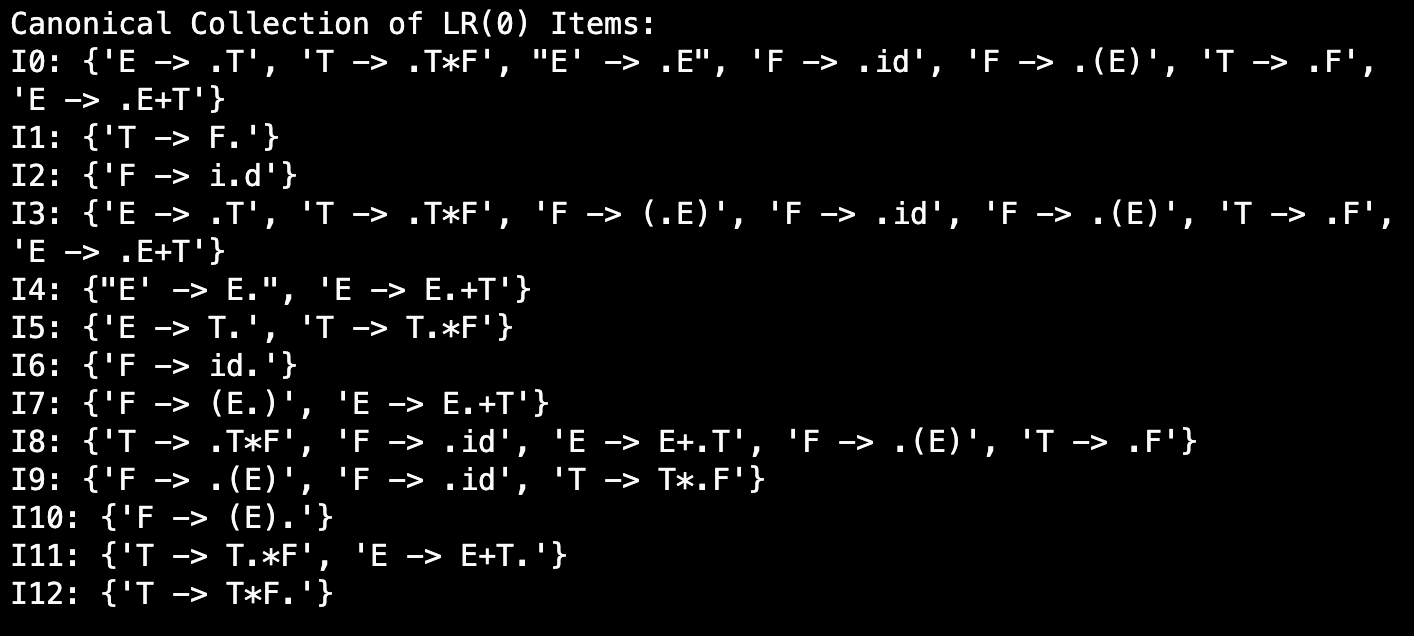
elif new\_state:

transitions[(i, symbol)] = new\_states.index(new\_state)

states = new\_states

return states, transitions

**Output**



**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)

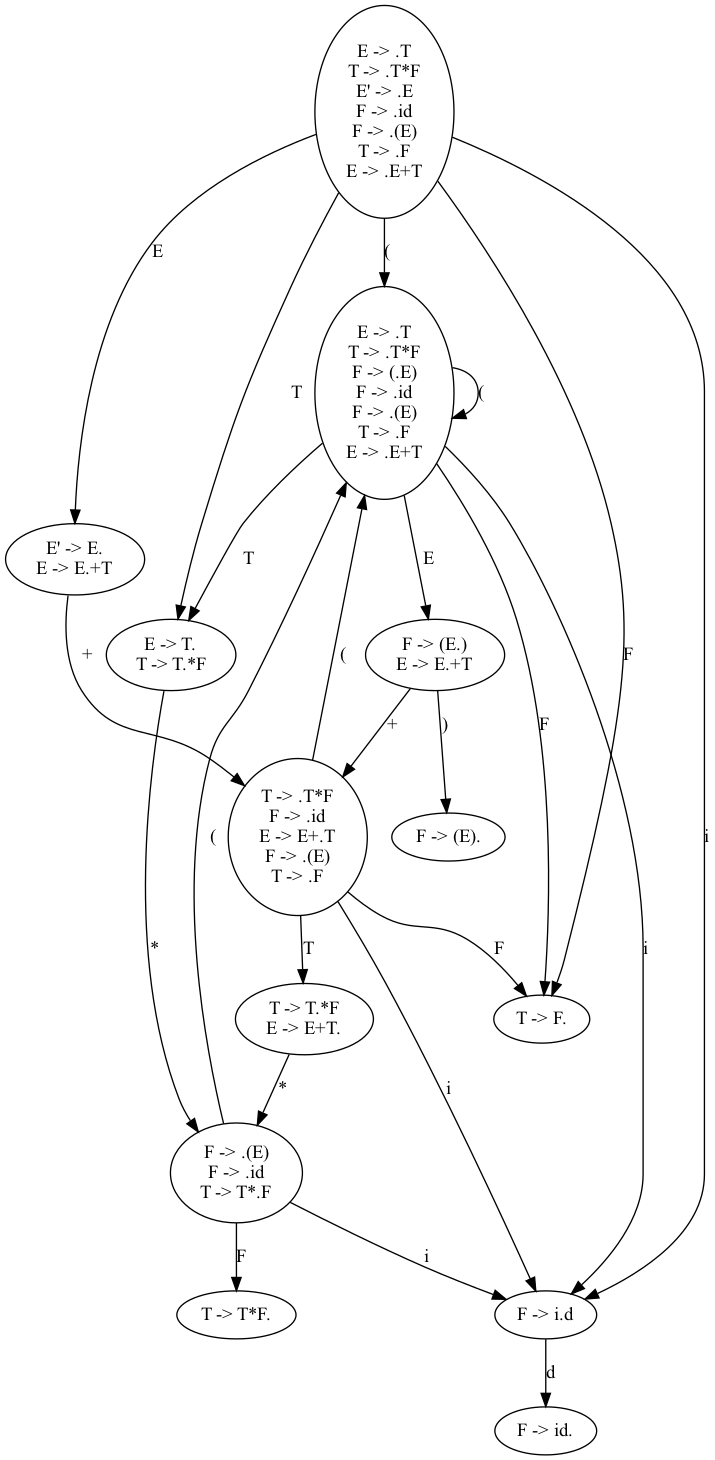
dot.node(f"I{i}", label=state\_label)

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**

****

**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:

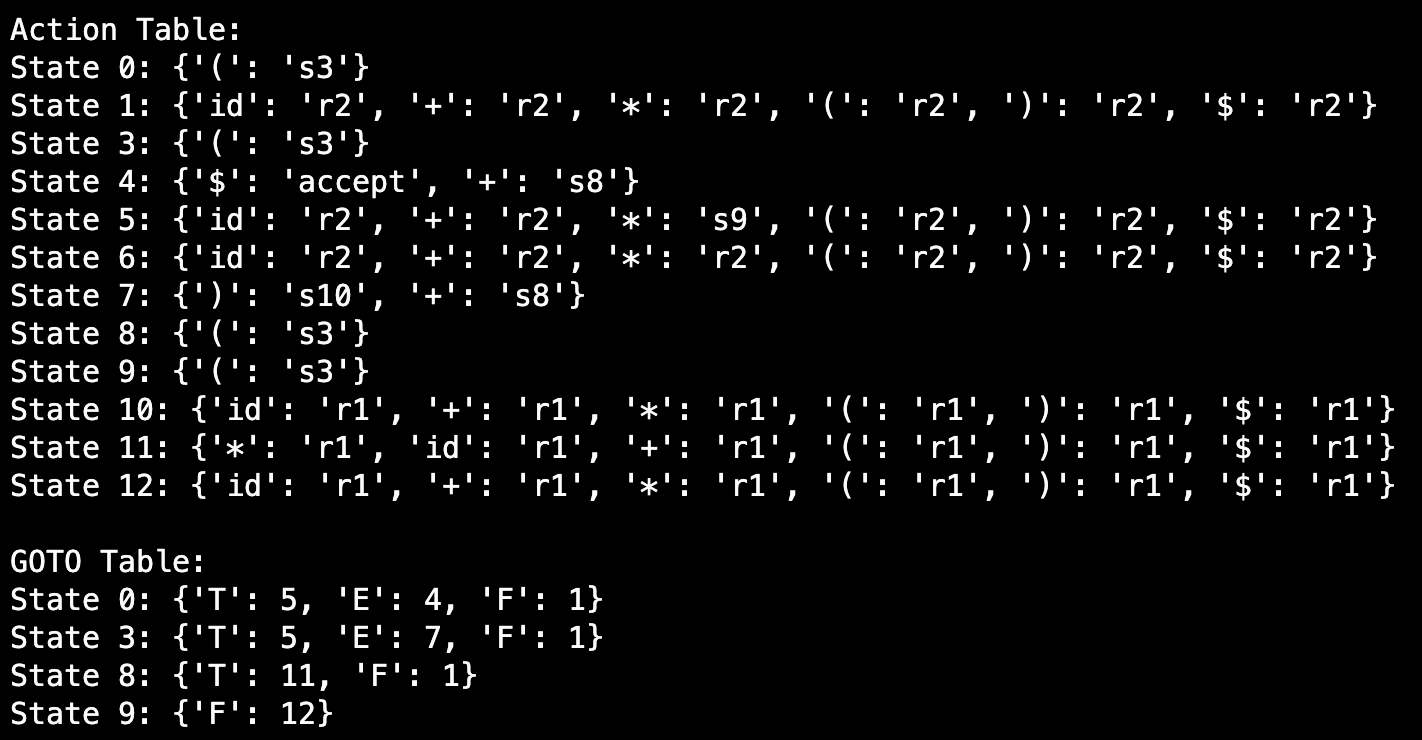
print(f"State {state}: {dict(action\_table[state])}")

print("\nGOTO Table:")

for state in goto\_table:

print(f"State {state}: {dict(goto\_table[state])}")

**Output**



**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**

