Practical 4

def read\_grammar():

grammar = {}

print("Enter grammar productions one per line in the format:")

print(" NonTerminal -> production1 | production2 | ...")

print("Use 'e' to denote the empty production. Enter an empty line when done.")

while True:

line = input().strip()

if not line:

break

if "->" not in line:

print("Invalid format. Please include '->'.")

continue

lhs, rhs = line.split("->")

lhs = lhs.strip()

if lhs not in grammar:

grammar[lhs] = []

alternatives = rhs.split("|")

for alt in alternatives:

alt = alt.strip()

if alt == "e":

grammar[lhs].append(["e"])

else:

grammar[lhs].append(alt.split())

return grammar

def compute\_first\_sets(grammar):

first = {nonterm: set() for nonterm in grammar}

changed = True

while changed:

changed = False

for nonterm, productions in grammar.items():

for prod in productions:

if prod == ["e"]:

if "e" not in first[nonterm]:

first[nonterm].add("e")

changed = True

else:

for symbol in prod:

if symbol not in grammar:

if symbol not in first[nonterm]:

first[nonterm].add(symbol)

changed = True

break

else:

before = len(first[nonterm])

first[nonterm].update(first[symbol] - {"e"})

if len(first[nonterm]) > before:

changed = True

if "e" not in first[symbol]:

break

return first

def first\_of\_string(symbols, first, grammar):

result = set()

for symbol in symbols:

if symbol not in grammar:

result.add(symbol)

return result

result.update(first[symbol] - {"e"})

if "e" not in first[symbol]:

return result

result.add("e")

return result

def compute\_follow\_sets(grammar, first, start\_symbol):

follow = {nonterm: set() for nonterm in grammar}

follow[start\_symbol].add("$")

changed = True

while changed:

changed = False

for A, productions in grammar.items():

for prod in productions:

for i, symbol in enumerate(prod):

if symbol in grammar:

before = len(follow[symbol])

beta = prod[i + 1:]

if beta:

beta\_first = first\_of\_string(beta, first, grammar)

follow[symbol].update(beta\_first - {"e"})

if "e" in beta\_first:

follow[symbol].update(follow[A])

else:

follow[symbol].update(follow[A])

if len(follow[symbol]) > before:

changed = True

return follow

def construct\_ll1\_table(grammar, first, follow):

"""Constructs the LL(1) parsing table."""

table = {nonterm: {} for nonterm in grammar}

def add\_entry(nonterm, terminal, production):

if terminal in table[nonterm]:

print(f"Conflict detected for table[{nonterm}, {terminal}]. Grammar may not be LL(1).")

table[nonterm][terminal] = production

for nonterm, productions in grammar.items():

for prod in productions:

prod\_first = first\_of\_string(prod, first, grammar)

for terminal in prod\_first - {"e"}:

add\_entry(nonterm, terminal, prod)

if "e" in prod\_first:

for terminal in follow[nonterm]:

add\_entry(nonterm, terminal, prod)

return table

def parse\_input\_string(input\_string, start\_symbol, table, grammar):

tokens = input\_string.split() + ["$"]

stack = ["$", start\_symbol]

index = 0

while stack:

top = stack.pop()

current = tokens[index]

if top not in grammar and top != "$":

if top == current:

index += 1

else:

print(f"Error: expected {top} but got {current}")

return False

elif top == "$":

return current == "$"

else:

if current in table[top]:

production = table[top][current]

if production != ["e"]:

stack.extend(reversed(production))

else:

print(f"No rule for nonterminal {top} with token {current}")

return False

return False

def main():

grammar = read\_grammar()

if not grammar:

print("No grammar entered. Exiting.")

return

start\_symbol = list(grammar.keys())[0]

print("\nGrammar:")

for nt, prods in grammar.items():

for prod in prods:

print(f" {nt} -> {' '.join(prod)}")

first = compute\_first\_sets(grammar)

follow = compute\_follow\_sets(grammar, first, start\_symbol)

print("\nFIRST sets:")

for nt in grammar:

print(f" {nt}: {first[nt]}")

print("\nFOLLOW sets:")

for nt in grammar:

print(f" {nt}: {follow[nt]}")

table = construct\_ll1\_table(grammar, first, follow)

print("\nLL(1) Parsing Table:")

for nt, rules in table.items():

for terminal, production in rules.items():

print(f" M[{nt}, {terminal}] = {nt} -> {' '.join(production)}")

input\_string = input("\nEnter an input string to parse (tokens separated by space), or press Enter to skip:\n").strip()

if input\_string:

if parse\_input\_string(input\_string, start\_symbol, table, grammar):

print("Input is accepted by the grammar.")

else:

print("Input is not accepted by the grammar.")

if \_\_name\_\_ == "\_\_main\_\_":

main()