**Lab-07**

**Unification in first order logic**

import re

def is\_variable(x):

"""A variable starts with a lowercase letter and has no parentheses."""

return isinstance(x, str) and x[0].islower() and '(' not in x and ')' not in x

def occur\_check(var, expr, theta):

"""Check if variable occurs inside expr (prevents infinite recursion)."""

if var == expr:

return True

elif isinstance(expr, list):

return any(occur\_check(var, sub, theta) for sub in expr)

elif expr in theta:

return occur\_check(var, theta[expr], theta)

return False

def substitute(expr, theta):

"""Apply substitution θ to expr recursively."""

if isinstance(expr, list):

return [substitute(e, theta) for e in expr]

elif expr in theta:

return substitute(theta[expr], theta)

else:

return expr

def unify(x, y, theta=None):

"""Main unification function."""

if theta is None:

theta = {}

if theta == "FAIL":

return "FAIL"

# Apply substitutions

x = substitute(x, theta)

y = substitute(y, theta)

if x == y:

return theta

elif is\_variable(x):

return unify\_var(x, y, theta)

elif is\_variable(y):

return unify\_var(y, x, theta)

elif isinstance(x, list) and isinstance(y, list):

# First element is the predicate / function symbol

if x[0] != y[0]:

return "FAIL" # predicate/function mismatch

if len(x) != len(y):

return "FAIL" # different arity

for xi, yi in zip(x[1:], y[1:]):

theta = unify(xi, yi, theta)

if theta == "FAIL":

return "FAIL"

return theta

else:

return "FAIL"

def unify\_var(var, x, theta):

"""Unify variable var with x."""

if var in theta:

return unify(theta[var], x, theta)

elif x in theta:

return unify(var, theta[x], theta)

elif occur\_check(var, x, theta):

return "FAIL"

else:

new\_theta = theta.copy()

new\_theta[var] = x

return new\_theta

# ---------------- Parsing Functions ----------------

def parse\_expression(expr\_str):

"""

Convert a FOL expression string into a nested list.

Example: "Knows(John,Father(x))" -> ['Knows', 'John', ['Father', 'x']]

"""

expr\_str = expr\_str.strip()

if '(' not in expr\_str:

return expr\_str

match = re.match(r'(\w+)\((.\*)\)', expr\_str)

if not match:

return expr\_str

functor = match.group(1)

args\_str = match.group(2)

# Split arguments, considering nested parentheses

args = []

depth = 0

current = ''

for ch in args\_str:

if ch == ',' and depth == 0:

args.append(parse\_expression(current.strip()))

current = ''

else:

if ch == '(':

depth += 1

elif ch == ')':

depth -= 1

current += ch

if current:

args.append(parse\_expression(current.strip()))

return [functor] + args

# ---------------- Main Program ----------------

def main():

print("==== First-Order Logic Unification ====")

expr1 = input("Enter first expression (e.g. Knows(John,x)): ").strip()

expr2 = input("Enter second expression (e.g. Knows(y,Bill)): ").strip()

x = parse\_expression(expr1)

y = parse\_expression(expr2)

result = unify(x, y)

print("\n--- Result ---")

if result == "FAIL":

print("The expressions CANNOT be unified.")

else:

print("Unifier found:")

for var, val in result.items():

print(f" {var} → {val}")

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

main()

