LAB-06

PROPOSITIONAL LOGIC

from itertools import product

# Function for logical implication

def imply(a, b):

return (not a) or b

# Convert user input to Python-evaluable expressions

import re

def convert\_to\_python\_syntax(s):

# Replace logical operators

s = s.replace('!','not ')

s = s.replace('∧',' and ')

s = s.replace('∨',' or ')

s = s.replace('↔','==')

# Replace -> with imply(a,b) properly

pattern = r'([A-Za-z() not]+)\s\*->\s\*([A-Za-z() not]+)'

while re.search(pattern, s):

s = re.sub(pattern, r'imply(\1, \2)', s)

return s

# Input KB

n = int(input("Enter number of KB sentences: "))

kb\_original = []

kb\_converted = []

for i in range(n):

s = input(f"Enter sentence {i+1} (use !, ∧, ∨, ->, ↔): ")

kb\_original.append(s)

kb\_converted.append(convert\_to\_python\_syntax(s))

# Input Queries

queries\_input = input("Enter queries separated by comma: ")

queries\_original = [q.strip() for q in queries\_input.split(',')]

queries\_converted = [convert\_to\_python\_syntax(q) for q in queries\_original]

# Extract all variables from KB and queries

import re

variables = set()

for sentence in kb\_original + queries\_original:

vars\_in\_s = re.findall(r'\b[A-Za-z]\b', sentence)

variables.update(vars\_in\_s)

variables = sorted(list(variables)) # sort for consistent order

# Generate all possible truth assignments

assignments = list(product([False, True], repeat=len(variables)))

# Print Truth Table Header

header = variables + kb\_original + ['KB True?']

print("\nTruth Table:")

print(' '.join(f"{h:<8}" for h in header))

print('-' \* (len(header) \* 10))

# Evaluate each row

models = [] # rows where KB is True

for vals in assignments:

valuation = dict(zip(variables, vals))

kb\_values = [eval(expr, {"imply": imply}, valuation) for expr in kb\_converted]

kb\_true = all(kb\_values)

if kb\_true:

models.append(valuation)

row = vals + tuple(kb\_values) + (kb\_true,)

print(' '.join(f"{str(x):<8}" for x in row))

# Entailment check

print("\nModels where KB is True:")

for m in models:

print(m)

print("\nEntailment results:")

for i, query in enumerate(queries\_converted):

entails = all(eval(query, {"imply": imply}, m) for m in models)

print(f"Does KB entail {queries\_original[i]}? --> {entails}")



