

Propositional Logic

Implementation of truth-table enumeration algorithm for deciding propositional entailment.

i.e., Create a knowledge base using propositional logic and show that the given query entails the knowledge base or not.

```
def is_variable(x):
    return isinstance(x, str) and x.islower()

def is_constant(x):
    return isinstance(x, str) and (x.isupper() or x.isdigit() or (x.isalpha() and not x.islower()))

def occurs_in(var, expr):
    if var == expr:
        return True
    if isinstance(expr, dict):
        return any(occurs_in(var, arg) for arg in expr.get('args', []))
    return False

def predicate_symbol(expr):
    if isinstance(expr, dict) and 'pred' in expr:
        return expr['pred']
    return None

def apply_substitution(subst, expr):
    if isinstance(expr, str):
        return subst.get(expr, expr)
    elif isinstance(expr, dict):
        return {
            'pred': expr['pred'],
            'args': [apply_substitution(subst, arg) for arg in expr.get('args', [])]
        }
    return expr

def compose_subst(s1, s2):
    result = {}
    for v, val in s1.items():
        result[v] = apply_substitution(s2, val)
    for v, val in s2.items():
        result[v] = val
    return result

def unify(x, y):
    if x == y:
        return {}
    if isinstance(x, str) and is_variable(x):
```

```

    if occurs_in(x, y):
        return "FAILURE"
    return {x: y}
if isinstance(y, str) and is_variable(y):
    if occurs_in(y, x):
        return "FAILURE"
    return {y: x}
if isinstance(x, str) and isinstance(y, str):
    return "FAILURE"
if isinstance(x, dict) and isinstance(y, dict):
    if predicate_symbol(x) != predicate_symbol(y):
        return "FAILURE"
    if len(x.get('args', [])) != len(y.get('args', [])):
        return "FAILURE"
    SUBST = {}
    for a, b in zip(x['args'], y['args']):
        a_ap = apply_substitution(SUBST, a)
        b_ap = apply_substitution(SUBST, b)
        S = unify(a_ap, b_ap)
        if S == "FAILURE":
            return "FAILURE"
        if S:
            SUBST = compose_subst(SUBST, S)
    return SUBST
return "FAILURE"

def sentence_to_str(sentence):
    if isinstance(sentence, str):
        return sentence
    elif isinstance(sentence, dict):
        args_str = ", ".join(sentence_to_str(arg) for arg in sentence.get('args', []))
        return f"{sentence['pred']}({args_str})"
    return str(sentence)

def str_to_sentence(s):
    s = s.strip()
    pred_end = s.find("(")
    if pred_end == -1:
        return s
    pred = s[:pred_end]
    args_str = s[pred_end+1:-1]
    args = [a.strip() for a in args_str.split(",")] if args_str else []
    return {'pred': pred, 'args': args}

def find_substitutions_for_premises(premises, known_facts):
    results = []
    def backtrack(i, subst):
        if i == len(premises):

```

```

        results.append(subst.copy())
    return
    prem = apply_substitution(subst, premises[i])
    for fact in known_facts:
        S = unify(prem, fact)
        if S == "FAILURE":
            continue
        new_subst = compose_subst(subst, S)
        backtrack(i+1, new_subst)
    backtrack(0, {})
    return results

def sentence_in_list(sentence, lst):
    s_str = sentence_to_str(sentence)
    return any(sentence_to_str(s) == s_str for s in lst)

def fol_fc_ask(KB, alpha):
    query = alpha
    known_facts = []
    agenda = []
    for premises, concl in KB:
        if not premises:
            fact = concl
            if not sentence_in_list(fact, known_facts):
                known_facts.append(fact)
                agenda.append(fact)
    while agenda:
        fact = agenda.pop(0)
        if unify(fact, query) != "FAILURE":
            return True
    for premises, concl in KB:
        subs = find_substitutions_for_premises(premises, known_facts)
        for s in subs:
            new_fact = apply_substitution(s, concl)
            if not sentence_in_list(new_fact, known_facts):
                known_facts.append(new_fact)
                agenda.append(new_fact)
    return False

KB = [
    ([], {'pred': 'prime', 'args': ['11']}),
    ([{'pred': 'prime', 'args': ['x']}, {'pred': 'odd', 'args': ['x']}]
]

alpha = {'pred': 'odd', 'args': ['11']}
result = fol_fc_ask(KB, alpha)
print("Result:", result)

```

OUTPUT:



Truth Table:

A	B	C	KB	alpha	KB \wedge α
True	True	True	True	True	True
True	True	False	False	True	False
True	False	True	True	True	True
True	False	False	True	True	True
False	True	True	True	True	True
False	True	False	False	False	False
False	False	True	False	True	False
False	False	False	False	False	False

Models where KB and alpha are true:

```
{'A': True, 'B': True, 'C': True}
{'A': True, 'B': False, 'C': True}
{'A': True, 'B': False, 'C': False}
{'A': False, 'B': True, 'C': True}
```

Does KB entail alpha? True