AI LAB

LAB 8

Create a knowledge base consisting of first order logic statements and prove the given query using forward reasoning.

CODE:

```
from copy import deepcopy

# Utility: standardize variables in a rule
def standardize_variables(rule, counter):
    """
    Replace all variables in a rule with new unique ones.
    """
    mapping = {}
    new_rule = deepcopy(rule)
    for term in new_rule['if'] + [new_rule['then']]:
        for i, arg in enumerate(term['args']):
            if isinstance(arg, str) and arg[0].islower():
                if arg not in mapping:
                    mapping[arg] = f"v{counter}"
                    counter += 1
                term['args'][i] = mapping[arg]
    return new_rule, counter


# Unification algorithm
def unify(x, y, subst=None):
    if subst is None:
        subst = {}
    if x == y:
        return subst
    elif isinstance(x, str) and x[0].islower():
        subst[x] = y
        return subst
    elif isinstance(y, str) and y[0].islower():
        subst[y] = x
        return subst
    elif isinstance(x, dict) and isinstance(y, dict) and x['pred'] ==
y['pred']:
        for a, b in zip(x['args'], y['args']):
            subst = unify(a, b, subst)
            if subst is None:
```

```
                return None
        return subst
    else:
        return None


# Apply substitution θ to an atom
def substitute(theta, atom):
    new_atom = deepcopy(atom)
    new_atom['args'] = [theta.get(arg, arg) for arg in atom['args']]
    return new_atom


# FOL-FC-ASK
def fol_fc_ask(KB, query):
    """
    KB: list of rules or facts
        - Facts: {'if': [], 'then': {'pred': 'P', 'args': ['a']}}
        - Rules: {'if': [{'pred': 'P', 'args': ['x']}, ...], 'then':
{'pred': 'Q', 'args': ['x']}}
    query: atomic sentence, e.g. {'pred': 'Q', 'args': ['a']}
    """
    new = []
    counter = 0
    inferred = set()

    while True:
        new = []
        for rule in KB:
            # Standardize variables
            std_rule, counter = standardize_variables(rule, counter)
            if_part = std_rule['if']
            then_part = std_rule['then']

            # For each combination of facts that could match the IF part
            matches = [{}] # Changed from [[]] to [{}] here
            for premise in if_part:
                new_matches = []
                for fact in KB:
                    if not fact['if']:  # atomic fact
                        # For each existing substitution in matches, try
to unify the premise with the fact
                        for m in matches:
                            # Create a combined substitution for the
current premise and fact
```

```
                            combined_m = m.copy() # Make a copy to avoid
modifying m directly in the loop
                            theta = unify(premise, fact['then'],
combined_m)
                            if theta is not None:
                                # Ensure theta is consistent with m (all
variables in m should map to same values)
                                is_consistent = True
                                for var, val in m.items():
                                    if var in theta and theta[var] != val:
                                        is_consistent = False
                                        break
                                if is_consistent:
                                    new_matches.append(theta)
                matches = new_matches

            # Infer new facts
            for theta in matches:
                new_fact = substitute(theta, then_part)
                if new_fact not in [f['then'] for f in KB if not f['if']]:
                    new.append({'if': [], 'then': new_fact})
                    KB.append({'if': [], 'then': new_fact})

                    # Check if query is satisfied
                    theta_q = unify(new_fact, query)
                    if theta_q is not None:
                        print("Query proven with substitution:", theta_q)
                        return True

        if not new:
            return False


# Example Knowledge Base
KB = [
    {'if': [], 'then': {'pred': 'Parent', 'args': ['John', 'Mary']}},
    {'if': [], 'then': {'pred': 'Parent', 'args': ['Mary', 'Sue']}},
    {'if': [{'pred': 'Parent', 'args': ['x', 'y']},
            {'pred': 'Parent', 'args': ['y', 'z']}],
     'then': {'pred': 'Grandparent', 'args': ['x', 'z']}}
]

query = {'pred': 'Grandparent', 'args': ['John', 'Sue']}

# Run the forward chaining
```

```
result = fol_fc_ask(KB, query)
print("Result:", result)
```

Output:

```
Query proven with substitution: {}
Result: True
```