```
def is_variable(term):
    """Check if a term is a variable."""
    return isinstance(term, str) and term.islower()

def apply_substitution(term, subst):
    """Apply a substitution to a term."""
    if is_variable(term) and term in subst:
        return apply_substitution(subst[term], subst)
    if isinstance(term, tuple):  # If term is a function, apply substitution to
        return (term[0], [apply_substitution(arg, subst) for arg in term[1]])
    return term  # Return the term as-is for constants or unbound variables

def unify(term1, term2, subst=None):
    """Unify two terms."""
    if subst is None:
        subst = {}
    if term1 == term2:
        return subst
    if is_variable(term1):
        return unify_variable(term1, term2, subst)
    if is_variable(term2):
        return unify_variable(term2, term1, subst)
    if isinstance(term1, tuple) and isinstance(term2, tuple):
        if term1[0] != term2[0] or len(term1[1]) != len(term2[1]):
            return None
        for arg1, arg2 in zip(term1[1], term2[1]):
            subst = unify(arg1, arg2, subst)
            if subst is None:
                return None
        return subst
    return None

def unify_variable(var, term, subst):
    """Unify a variable with a term."""
    if var in subst:
        return unify(subst[var], term, subst)
    if occurs_check(var, term, subst):
        return None
    subst[var] = term
    return subst

def occurs_check(var, term, subst):
    """Check if a variable occurs in a term."""
    if var == term:
        return True
    if isinstance(term, tuple):
        return any(occurs_check(var, arg, subst) for arg in term[1])
    if var in subst and occurs_check(var, subst[var], subst):
        return True
    return False

def fact_in_kb(fact, kb):
    """
    Check if a fact is already in the knowledge base.
    Args:
        fact: The fact to check.
        kb: The list of known facts.
    Returns:
        True if the fact is in the knowledge base, otherwise False.
    """
    for known_fact in kb:
        if unify(fact, known_fact) is not None:
            return True
    return False

def forward_reasoning(kb, query):
    """
    Perform forward reasoning on the knowledge base (KB) to prove the query.
    Args:
        kb: The knowledge base, a list of first-order logic rules or facts.
        query: The goal to prove.
    Returns:
        True if the query can be proved, otherwise False.
    """
    known_facts = []  # Store known facts as a list
    new_facts = True

    while new_facts:
        new_facts = False

        for rule in kb:
            if isinstance(rule, tuple) and rule[0] == "implies":  # Implication
                conditions, conclusion = rule[1], rule[2]

                substitutions = [{}]
                for condition in conditions:
                    next_substitutions = []
                    for fact in known_facts:
                        subst = unify(condition, fact)
                        if subst is not None:
                            next_substitutions.append(subst)
                    substitutions = [
                        {**s1, **s2} for s1 in substitutions for s2 in next_sub
                    ]

                for subst in substitutions:
```

```
                    derived_fact = apply_substitution(conclusion, subst)
                    if not fact_in_kb(derived_fact, known_facts):
                        known_facts.append(derived_fact)
                        new_facts = True

            else:  # It's a fact
                if not fact_in_kb(rule, known_facts):
                    known_facts.append(rule)
                    new_facts = True

        # Check if the query is in the known facts
        for fact in known_facts:
            if unify(fact, query) is not None:
                return True

    return False


# Example Usage
if __name__ == "__main__":
    # Knowledge Base
    kb = [
        ("implies", [("human", ["x"])], ("mortal", ["x"])),  # human(x) -> mort
        ("human", ["socrates"]),  # human(socrates)
    ]

    # Query
    query = ("mortal", ["socrates"])  # Is Socrates mortal?

    # Perform forward reasoning
    result = forward_reasoning(kb, query)

    if result:
        print(f"The query {query} is true based on the knowledge base.")
    else:
        print(f"The query {query} cannot be proved from the knowledge base.")
```

```
The query ('mortal', ['socrates']) is true based on the knowledge base.
```