Create a knowledge base consisting of FOL statements and prove the given query using forward reasoning

Python Code:

```
import re

class ForwardReasoning:
    def __init__(self, rules, facts):
        self.rules = rules  # List of rules (condition -> result)
        self.facts = set(facts)  # Known facts
    def match_condition(self, condition):
        variable_map = {}
        for cond in condition:
            if "∀" in cond:  # Universal quantifier handling
                var = cond[2:-1].strip()  # Extract variable from ∀x
                for fact in self.facts:
                    if var in fact:
                        variable_map[var] = fact
                        break
                else:
                    return False, variable_map
            elif "∃" in cond:  # Existential quantifier handling
                var = cond[2:-1].strip()  # Extract variable from ∃x
                for fact in self.facts:
                    if var in fact:
                        variable_map[var] = fact
                        return True, variable_map
```

```
            return False, variable_map
        else:  # Simple fact match
            fact_match = False
            for fact in self.facts:
                if self.match_fact(cond, fact, variable_map):
                    fact_match = True
                    break
            if not fact_match:
                return False, variable_map
    return True, variable_map

def match_fact(self, cond, fact, variable_map):

    var_pattern = re.compile(r'\b[a-zA-Z]+\b')

    condition_parts = re.findall(var_pattern, cond)
    fact_parts = re.findall(var_pattern, fact)

    if len(condition_parts) == len(fact_parts):
        for var, fact_part in zip(condition_parts, fact_parts):
            if var not in variable_map:
                variable_map[var] = fact_part
            elif variable_map[var] != fact_part:
                return False
        return True
```

```
        return cond == fact  # If not variable-based, check exact match


    def infer(self, query):
        """
        Forward chaining algorithm to infer if the query can be derived from rules and facts.
        """
        applied_rules = True

        while applied_rules:
            applied_rules = False
            for condition, result in self.rules:
                matched, variable_map = self.match_condition(condition)
                if matched and result not in self.facts:
                    self.facts.add(result)  # Add the result to known facts
                    applied_rules = True
                    print(f"Applied rule: {condition} -> {result}")

                    # If the query is inferred, return True immediately
                    if self.match_fact(query, result, variable_map):
                        return True

        # Return True if the query is in facts after the reasoning process, else False
        return self.match_fact(query, result, variable_map)
```

```
def get_input_rules():
    rules = []
    while True:
        rule = input("Enter rule (or 'done' to finish): ").strip()
        if rule.lower() == "done":
            break

        # Parse the rule properly
        if "=>" in rule:
            # Check for complex expressions with quantifiers and split the rule
            condition_str, result = rule.split("=>")

            # Remove extra spaces and deal with complex conditions
            condition_str = condition_str.strip()
            result = result.strip()

            # Handle potential multiple conditions (ANDs)
            conditions = set(re.split(r'\s*AND\s*', condition_str))

            # Add the rule to the list
            rules.append((conditions, result))

    return rules

def get_input_facts():
```

```
    facts = set()
    while True:
        fact = input("Enter fact (or 'done' to finish): ").strip()
        if fact.lower() == "done":
            break
        facts.add(fact)
    return facts

def get_input_query():
    query = input("Enter the query: ").strip()
    return query

# Main program to run the forward reasoning
def main():
    print("Enter the rules:")
    rules = get_input_rules()

    print("\nEnter the facts:")
    facts = get_input_facts()

    print("\nEnter the query:")
    query = get_input_query()

    # Initialize and run forward reasoning
    reasoner = ForwardReasoning(rules, facts)
```

```
    result = reasoner.infer(query)

    # Debugging Output
    print("\nFinal facts:")
    print(reasoner.facts)
    print(f"\nQuery '{query}' inferred: {result}")

# Call the main function to start
main()
```

Output:

Enter the rules:

Enter rule (or 'done' to finish): American(p) AND Weapon(q) AND Sells(p, q, r) AND Hostile(r) => Criminal(p)

Enter rule (or 'done' to finish): ∃x (Owns(A, x) AND Missile(x)) => Missile(x) AND Weapon(x)

Enter rule (or 'done' to finish): ∀x(Missile(x) AND Owns(A, x)) => Sells(Robert, x, A)

Enter rule (or 'done' to finish): Missile(x) => Weapon(x)

Enter rule (or 'done' to finish): ∀x (Enemy(x, America)) => Hostile(x)

Enter rule (or 'done' to finish): done

Enter the facts:

Enter fact (or 'done' to finish): American(Robert)

Enter fact (or 'done' to finish): Enemy(A, America)

Enter fact (or 'done' to finish): Owns(A, T1)

Enter fact (or 'done' to finish): Missile(T1)

Enter fact (or 'done' to finish): done

Enter the query:

Enter the query: Criminal(Robert)

Applied rule: {'Missile(x)'} -> Weapon(x)

Final facts:

{'American(Robert)', 'Owns(A, T1)', 'Missile(T1)', 'Enemy(A, America)', 'Weapon(x)'}

Query 'Criminal(Robert)' inferred: True
