Knowledge Base -Resolution:

```
# Function to negate a literal
def negate(literal):
    """Negate a literal."""
    if isinstance(literal, tuple) and literal[0] == "not":
        # If the literal is already negated, return the positive form
        return literal[1]
    else:
        # Otherwise, return the negated form
        return ("not", literal)

# Function to resolve two clauses
def resolve(clause1, clause2):
    """Return the resolvent of two clauses."""
    resolvents = set()
    for literal1 in clause1:
        for literal2 in clause2:
            if literal1 == negate(literal2):
                resolvent = (clause1 - {literal1}) | (clause2 -
{literal2})
                print(f" Resolving literal: {literal1} with {literal2}")
                print(f" Resulting Resolvent: {resolvent}")
                resolvents.add(frozenset(resolvent))
    return resolvents

# Function to perform resolution on the KB and query with detailed output
def resolution_algorithm(KB, query):
    """Perform the resolution algorithm to check if the query can be
proven."""
    print("\n--- Step-by-Step Resolution Process ---")
    # Add the negation of the query to the knowledge base
    negated_query = negate(query)
    KB.append(frozenset([negated_query]))
    print(f"Negated Query Added to KB: {negated_query}")

    # Initialize the set of clauses to process
    clauses = set(KB)

    step = 1
    while True:
```

```
        new_clauses = set()
        print(f"\nStep {step}: Resolving Clauses")
        for c1 in clauses:
            for c2 in clauses:
                if c1 != c2:
                    print(f" Resolving clauses: {c1} and {c2}")
                    resolvent = resolve(c1, c2)

                    for res in resolvent:
                        if frozenset([]) in resolvent:
                            print("\nEmpty clause derived! The query is
provable.")
                            return True  # Empty clause found,
contradiction, query is provable

                        new_clauses.add(res)

        if new_clauses.issubset(clauses):
            print("\nNo new clauses can be derived. The query is not
provable.")
            return False  # No new clauses, query is not provable

        clauses.update(new_clauses)
        step += 1

# Knowledge Base (KB) from the image facts
KB = [
    frozenset([("not", "food(x)"), ("likes", "John", "x")]),  # 1
    frozenset([("food", "Apple")]),  # 2
    frozenset([("food", "vegetables")]),  # 3
    frozenset([("not", "eats(y, z)"), ("killed", "y"), ("food", "z")]),  #
4
    frozenset([("eats", "Anil", "Peanuts")]),  # 5
    frozenset([("alive", "Anil")]),  # 6
    frozenset([("not", "eats(Anil, w)"), ("eats", "Harry", "w")]),  # 7
    frozenset([("killed", "g"), ("alive", "g")]),  # 8
    frozenset([("not", "alive(k)"), ("not", "killed(k)")]),  # 9
    frozenset([("likes", "John", "Peanuts")])  # 10
]
```

```python
# Query to prove
query = ("likes", "John", "Peanuts")

# Perform resolution to check if the query is provable
result = resolution_algorithm(KB, query)
if result:
    print("\nQuery is provable.")
else:
    print("\nQuery is not provable.")
name = "Varsha Prasanth"
usn = "1BM22CS321"
print(f"Name: {name}, USN: {usn}")
```

OUTPUT

```
Resolving clauses: frozenset({('eats', 'Anil', 'Peanuts')}) and frozenset({('not', 'food(x)'), ('likes', 'John', 'x')})
Resolving clauses: frozenset({('killed', 'g'), ('alive', 'g')}) and frozenset({('alive', 'Anil')})
Resolving clauses: frozenset({('killed', 'g'), ('alive', 'g')}) and frozenset({('eats', 'Anil', 'Peanuts')})
Resolving clauses: frozenset({('killed', 'g'), ('alive', 'g')}) and frozenset({('food', 'Apple')})
Resolving clauses: frozenset({('killed', 'g'), ('alive', 'g')}) and frozenset({('not', ('likes', 'John', 'Peanuts'))})
Resolving clauses: frozenset({('killed', 'g'), ('alive', 'g')}) and frozenset({('eats', 'Harry', 'w'), ('not', 'eats(Anil, w)')})
Resolving clauses: frozenset({('killed', 'g'), ('alive', 'g')}) and frozenset({('food', 'vegetables')})
Resolving clauses: frozenset({('killed', 'g'), ('alive', 'g')}) and frozenset({('not', 'killed(k)'), ('not', 'alive(k)')})
Resolving clauses: frozenset({('killed', 'g'), ('alive', 'g')}) and frozenset({('not', 'eats(y, z)'), ('food', 'z'), ('killed', 'y')})
Resolving clauses: frozenset({('killed', 'g'), ('alive', 'g')}) and frozenset({('likes', 'John', 'Peanuts')})
Resolving clauses: frozenset({('killed', 'g'), ('alive', 'g')}) and frozenset({('not', 'food(x)'), ('likes', 'John', 'x')})
Resolving clauses: frozenset({('food', 'Apple')}) and frozenset({('alive', 'Anil')})
Resolving clauses: frozenset({('food', 'Apple')}) and frozenset({('eats', 'Anil', 'Peanuts')})
Resolving clauses: frozenset({('food', 'Apple')}) and frozenset({('killed', 'g'), ('alive', 'g')})
Resolving clauses: frozenset({('food', 'Apple')}) and frozenset({('not', ('likes', 'John', 'Peanuts'))})
Resolving clauses: frozenset({('food', 'Apple')}) and frozenset({('eats', 'Harry', 'w'), ('not', 'eats(Anil, w)')})
Resolving clauses: frozenset({('food', 'Apple')}) and frozenset({('food', 'vegetables')})
Resolving clauses: frozenset({('food', 'Apple')}) and frozenset({('not', 'killed(k)'), ('not', 'alive(k)')})
Resolving clauses: frozenset({('food', 'Apple')}) and frozenset({('not', 'eats(y, z)'), ('food', 'z'), ('killed', 'y')})
Resolving clauses: frozenset({('food', 'Apple')}) and frozenset({('likes', 'John', 'Peanuts')})
Resolving clauses: frozenset({('food', 'Apple')}) and frozenset({('not', 'food(x)'), ('likes', 'John', 'x')})
Resolving clauses: frozenset({('not', ('likes', 'John', 'Peanuts'))}) and frozenset({('alive', 'Anil')})
Resolving clauses: frozenset({('not', ('likes', 'John', 'Peanuts'))}) and frozenset({('eats', 'Anil', 'Peanuts')})
Resolving clauses: frozenset({('not', ('likes', 'John', 'Peanuts'))}) and frozenset({('killed', 'g'), ('alive', 'g')})
Resolving clauses: frozenset({('not', ('likes', 'John', 'Peanuts'))}) and frozenset({('food', 'Apple')})
Resolving clauses: frozenset({('not', ('likes', 'John', 'Peanuts'))}) and frozenset({('eats', 'Harry', 'w'), ('not', 'eats(Anil, w)')})
Resolving clauses: frozenset({('not', ('likes', 'John', 'Peanuts'))}) and frozenset({('food', 'vegetables')})
Resolving clauses: frozenset({('not', ('likes', 'John', 'Peanuts'))}) and frozenset({('not', 'killed(k)'), ('not', 'alive(k)')})
Resolving clauses: frozenset({('not', ('likes', 'John', 'Peanuts'))}) and frozenset({('not', 'eats(y, z)'), ('food', 'z'), ('killed', 'y')})
Resolving clauses: frozenset({('not', ('likes', 'John', 'Peanuts'))}) and frozenset({('likes', 'John', 'Peanuts')})
Resolving literal: ('not', ('likes', 'John', 'Peanuts')) with ('likes', 'John', 'Peanuts')
Resulting Resolvent: frozenset()

Empty clause derived! The query is provable.

Query is provable.
Name: Varsha Prasanth, USN: 1BM22CS321
```