Implementation of First Order Logic Using Resolution

```
# Knowledge Base (KB)
facts = {
"Eats(Anil, Peanuts)": True,
"not Killed(Anil)": True,
"Food(Apple)": True,
"Food(Vegetables)": True,
}

rules = [
# Rule: Food(X) :- Eats(Y, X) and not Killed(Y)
{"conditions": ["Eats(Y, X)", "not Killed(Y)"], "conclusion": "Food(X)"},
# Rule: Likes(John, X) :- Food(X)
{"conditions": ["Food(X)"], "conclusion": "Likes(John, X)"},
]

# Query
query = "Likes(John, Peanuts)"

# Helper function to substitute variables in a rule
def substitute(rule_part, substitutions):
for var, value in substitutions.items():
rule_part = rule_part.replace(var, value)
return rule_part
```

```
# Function to resolve the query
def resolve_query(facts, rules, query):
working_facts = facts.copy()
while True:
new_facts_added = False
for rule in rules:
conditions = rule["conditions"]
conclusion = rule["conclusion"]

# Try all substitutions for variables (X, Y) in the rules
for entity in ["Apple", "Vegetables", "Peanuts", "Anil", "John"]:
substitutions = {"X": "Peanuts", "Y": "Anil"}  # Fixed for this problem
resolved_conditions = [substitute(cond, substitutions) for cond in conditions]
resolved_conclusion = substitute(conclusion, substitutions)

# Check if all conditions are true
if all(working_facts.get(cond, False) for cond in resolved_conditions):
if resolved_conclusion not in working_facts:
working_facts[resolved_conclusion] = True
new_facts_added = True
print(f"Derived Fact: {resolved_conclusion}")

if not new_facts_added:
break
```

```
# Check if the query is resolved
return working_facts.get(query, False)

# Run the resolution process
if resolve_query(facts, rules, query):
print(f"Proven: {query}")
else:
print(f"Not Proven: {query}")
```

Output:

Derived Fact: Food(Peanuts)

Derived Fact: Likes(John, Peanuts)

Proven: Likes(John, Peanuts)