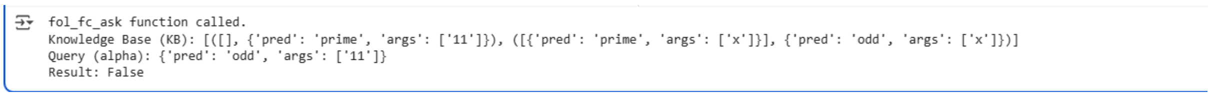
Forward Reasoning Algorithm

```
def unify(x, y):
    # A simple unifier (can be replaced by your detailed unify
function)
    if x == y:
        return {}
    if isinstance(x, str) and x.islower():
        return {x: y}
    if isinstance(y, str) and y.islower():
        return {y: x}
    return "FAILURE"

def apply_substitution(subst, sentence):
    if isinstance(sentence, str):
        return subst.get(sentence, sentence)
    elif isinstance(sentence, dict):
        return {
            'pred': sentence['pred'],
            'args': [apply_substitution(subst, arg) for arg in
sentence.get('args', [])]
        }
    return sentence

def sentence_to_str(sentence):
    if isinstance(sentence, str):
        return sentence
    elif isinstance(sentence, dict):
        args_str = ",".join(sentence_to_str(arg) for arg in
sentence.get('args', []))
        return f"{sentence['pred']}({args_str})"
    return str(sentence)

def str_to_sentence(s):
    # Very basic parser assuming format pred(arg1,arg2,...)
    pred_end = s.find("(")
    if pred_end == -1:
        return s
    pred = s[:pred_end]
    args_str = s[pred_end+1:-1]
    args = args_str.split(",") if args_str else []
    return {'pred': pred, 'args': args}

def sentence_in_KB_or_new(sentence, KB, new):
    s_str = sentence_to_str(sentence)
    for rule in KB:
        _, concl = rule
        if sentence_to_str(concl) == s_str:
                return True
        if s_str in new:
            return True
        return False

def find_substitutions_for_premises(premises, KB):
    # For demo: if no premises, return [{}] (empty substitution)
    if not premises:
        return [{}]
    # Otherwise, just return empty for simplicity
    return [{}]

# Placeholder for the fol_fc_ask function
def fol_fc_ask(KB, alpha):
    """
    A placeholder function for forward chaining inference.
    Replace with your actual implementation.
    """
    print("fol_fc_ask function called.")
    print("Knowledge Base (KB):", KB)
    print("Query (alpha):", alpha)
    # This is where your forward chaining logic would go.
    # For demonstration, let's assume it returns False.
    return False


# Example knowledge base: list of (premises, conclusion)
KB = [
    ([], {'pred': 'prime', 'args': ['11']}),  # Fact: prime(11)
    ([{'pred': 'prime', 'args': ['x']}], {'pred': 'odd', 'args':
['x']})  # Rule: prime(x) => odd(x)
]

alpha = {'pred': 'odd', 'args': ['11']}  # Query: odd(11)

result = fol_fc_ask(KB, alpha)

print("Result:", result)
```

Output:



```
fol_fc_ask function called.
Knowledge Base (KB): [([], {'pred': 'prime', 'args': ['11']}), ([{'pred': 'prime', 'args': ['x']}], {'pred': 'odd', 'args': ['x']})]
Query (alpha): {'pred': 'odd', 'args': ['11']}
Result: False
```