WEEK-6

Create a knowledgebase using prepositional logic and show that the given query entails the knowledge base or not

```
def evaluate_first(premise, conclusion):
    # Create all possible models for the variables p, q, and r
    models = [
        {'p': False, 'q': False, 'r': False},
        {'p': False, 'q': False, 'r': True},
        {'p': False, 'q': True, 'r': False},
        {'p': False, 'q': True, 'r': True},
        {'p': True, 'q': False, 'r': False},
        {'p': True, 'q': False, 'r': True},
        {'p': True, 'q': True, 'r': False},
        {'p': True, 'q': True, 'r': True}
    ]

    # Check if the premise logically entails the conclusion
    entails = True  # Initially assume the premise entails the conclusion
    for model in models:
        if evaluate_expression(premise, model) != evaluate_expression(conclusion, model):
            entails = False  # If any model disagrees, premise does not entail conclusion
            break

    # Return the result
    return entails
```

```
# Define a function to evaluate logical expressions for the second input
def evaluate_second(premise, conclusion):
    # Generate all possible truth assignments to p, q, and r
    models = [
        {'p': p, 'q': q, 'r': r}
        for p in [True, False]
        for q in [True, False]
        for r in [True, False]
    ]

    # Check if the conclusion holds in every model where the premise is true
    entails = all(evaluate_expression(premise, model) for model in models if evaluate_expression(conclusion, model) and evaluate_expression(premise, model))

    # Return the result
    return entails

# Define a function to evaluate logical expressions based on the given expression and model
def evaluate_expression(expression, model):
    # Evaluate the logical expression recursively using the given model
    if isinstance(expression, str):
        # Base case: if it's a single variable or literal
        return model.get(expression)
    elif isinstance(expression, tuple):
        op = expression[0]  # Get the operator
        if op == 'not':
```

```
        return not evaluate_expression(expression[1], model)  # Negation
    elif op == 'and':
        return evaluate_expression(expression[1], model) and evaluate_expression(expression[2], model)  # Conjunction
    elif op == 'if':
        return (not evaluate_expression(expression[1], model)) or evaluate_expression(expression[2], model)  # Implication
    elif op == 'or':
        return evaluate_expression(expression[1], model) or evaluate_expression(expression[2], model)  # Disjunction (OR)

# Premise and conclusion for the first and second inputs
first_premise = ('and', ('or', 'p', 'q'), ('or', ('not', 'r'), 'p')) # (p OR q) AND (NOT r OR p)
first_conclusion =  ('and', 'p', 'r')# p AND r

second_premise = ('and', ('or', ('not', 'q'), ('not', 'p'), 'r'), ('and', ('not', 'q'), 'p'), 'q')
second_conclusion = 'r'

# Evaluate both inputs using the merged function
result_first = evaluate_first(first_premise, first_conclusion)
result_second = evaluate_second(second_premise, second_conclusion)

# Print the results for both inputs
if result_first:
    print("For the first input: The knowledge base entails the query.")
else:
    print("For the first input: The knowledge base does not entail the query.")
```

```
if result_second:
    print("For the second input: The knowledge base entails the query.")
else:
    print("For the second input: The knowledge base does not entail the query.")
```

OUTPUT:

```
>>>
============ RESTART: C:/Users/Admin/Desktop/1BM21CS205 AI LAB/p5.py ===========
For the first input: The knowledge base does not entail the query.
For the second input: The knowledge base entails the query.
>>>
```