Logical Inference Utilizing Backward Chaining (First-Order Logic) Tutorial

Tutorial General Overview:.

The aim of Task 7 was to investigate and develop a **backward chaining inference engine for First-Order Logic (FOL)**. Backward chaining is a goal-oriented inference technique that involves tracing backward from a hypothesis to recursively identify the facts or subgoals that need to be satisfied in order for the hypothesis to be true, based on a set of logical rules.

In this task, I developed a backward chaining algorithm that builds an **AND/OR goal tree** to represent all the possible logical paths that can be taken to support a hypothesis. The system allows **First-Order Logic variables**, rule unification, and recursion.

Core Basis of FOL Functionality:

The main component of this task is the function: backchain_to_goal_tree(rules, hypothesis)

For this function:

1. Takes a **hypothesis** (a string representing a logical goal)
2. Takes a list of **IF-THEN rules**
3. Recursively searches for rules whose consequents match the hypothesis
4. Uses **unification** to bind variables when necessary
5. Builds an **AND/OR tree** of subgoals that must be satisfied
6. Returns a simplified goal tree representing all possible proofs

The leaves of the tree are atomic statements (strings), and internal nodes are either:

- **AND**: all subgoals must be satisfied
- **OR**: at least one subgoal path may satisfy the hypothesis

To improve interpretability, I also implemented a helper function pretty() that prints the AND/OR tree in a human-readable, indented format.

How FOL Backward Chaining Conceptually Works (background info):

Backward chaining is a top-down process:

1. Begin with a hypothesis (for example, "opus is a penguin")
2. Find the rules which might lead to this hypothesis
3. Substitute the hypothesis with the antecedent of the rule
4. Continue recursively until only basic facts are left

Backward chaining is very conceptually different from forward chaining because the latter derives conclusions from known facts. Backward chaining is particularly helpful when only specific questions need to be answered. Both chains are useful in different circumstances and should be used accordingly.

Testing 1 - Grounded (Concrete) Hypothesis:

- I first tested the system with a concrete hypothesis: "opus is a penguin"
- This verifies that the system correctly expands known rules and builds a valid explanation tree using constants.

Testing 2 - FOL (Variable-based) Hypothesis:

To confirm and verify the support for First-Order Logic, I executed a query that is variable-based: "(?x) is a penguin"

This query confirms that the system:

- **Unifies** correctly
- Handles symbolic variables correctly
- Builds a general goal tree that is valid for any entity

Example Goal Tree Output (Simplified Version):

```
 OR
   (?x) is a penguin
   AND
     OR
       (?x) is a bird
       (?x) has feathers
       (?x) flies
     (?x) does not fly
     (?x) swims
     (?x) is black and white
```

This tree illustrates that in order to prove an object is a penguin, one must not only prove all the defining characteristics of a penguin, but also demonstrate and elaborate on how being a bird can be proved.

Utilized Command Lines:

Running the overall backward chaining system:

```
python3 run_task7.py
```

Testing backward chaining interactively:

```
 python3
```

```
from data import zookeeper_rules
from lab1 import backchain_to_goal_tree
print(backchain_to_goal_tree(zookeeper_rules, "opus is a penguin"))
```

- The other 3 lines after “python3” is done inside the python3 system

Testing the variable-based First-Order Logic inference:

```
python3 - << 'PY'
from data import zookeeper_rules
from lab1 import backchain_to_goal_tree
from run_task7 import pretty

print(pretty(backchain_to_goal_tree(zookeeper_rules, "(?x) is a penguin")))
PY
```

- These commands validate both concrete and symbolic backward chaining behavior.

Summary Conclusion:

In this task for the AIEA Lab, I was able to successfully design and implement a backward chaining inference engine for First-Order Logic (FOL). The program has the ability to handle variable unification, recursive goal expansion, and the creation of AND/OR goal trees. Through a series of tests, including tests involving variables, I was able to verify and fully confirm that the program is able to correctly execute logical inference and generate meaningful reasoning structures.