HRISHIKESH TIWARI: CS22M047
ASHISH PARJAPATI: CS22M022

CS6770: Knowledge Representation and Reasoning

Problem 5: Backward Chaining (Group C)

1. Problem Statement:

Implement a Prolog-like backward chaining system using a depth-first strategy. Accept rules from a text file (as per the language specification) and convert them into XML. Accept the goal from a similar file. Show the final proof for the goal (if possible, graphically as a tree). One should be able to opt for more than one solution on giving a goal. Allow the use of Cut. The algorithm should output either a substitution or FALSE, along with a trace in a text file.

Input Format:

input.txt -- XML Parser -- input.xml
query.txt -- XML Parser -- query.xml

Output:

The algorithm should output either a substitution or FALSE, along with a trace in a text file.

2. Workflow:

1. We have placed the input horn-clause text file inside the "input" folder.
2. Then Run: `./run.sh` on the terminal inside the "krr" folder.
3. The output XML file has been generated in the "output" folder.
4. Same Procedure has been followed for the query text file.

5. Now inside the "code" folder run the following command on the terminal:

`python3 query_solver.py -k ../output/HC.xml -q ../output/HC_queries_sibling.xml`
for running queries for 'sibling' in the HC file.

Terminal File Generated:

```
(base) Hrishi@Hrishi-MacBook-Air code % python3 executer.py -k ../output/HC.xml -q ../output/HC_queries_sibling.xml
hasSibling(X200, Y199)


X = lucy
Y = jack
Enter c to look for more solutions:c
X = jack
Y = lucy
Enter c to look for more solutions:c
True
(base) Hrishi@Hrishi-MacBook-Air code %
```

Trace File Generated:

```
FINISHED PARSING KB
FINISHED PARSING QUERY
Query:
hasSibling(X68, Y69)


Depth: 0
Solving goal: hasSibling(X68, Y69)
Depth: 0
goal: hasSibling(X68, Y69)
match: hasSibling(lucy, jack)
Unifier:
X68 :lucy
Y69 :jack


All Goals Solved!!


MGU:
X68 : lucy
Y69 : jack


X = lucy
Y = jack
User chose to continue
Depth: 0
goal: hasSibling(X68, Y69)
match: hasSibling(jack, lucy)
Unifier:
X68 :jack
Y69 :lucy


All Goals Solved!!


MGU:
X68 : jack
Y69 : lucy


X = jack
Y = lucy
User chose to continue
True
```

6. Now inside the "code" folder run the following command on the terminal:

`python3 query_solver.py -k ../output/HC.xml -q ../output/HC_queries_append.xml`
for running queries for 'append' in the HC file.

Terminal File Generated:

```
(base) Hrishi@Hrishi-MacBook-Air code % python3 executer.py -k ../output/HC.xml -q ../output/HC_queries_append.xml
append(cons(1, cons(2, [])), cons(3, cons(4, [])), C199)


C = [1, 2, 3, 4]
Enter c to look for more solutions:c
True
```

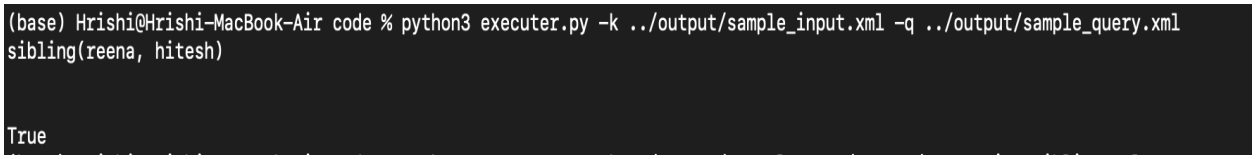
Trace File Generated:

```
FINISHED PARSING KB
FINISHED PARSING QUERY
Query:
append(cons(1, cons(2, [])), cons(3, cons(4, [])), C109)


Depth: 0
Solving goal: append(cons(1, cons(2, [])), cons(3, cons(4, [])), C109)
Depth: 0
goal: append(cons(1, cons(2, [])), cons(3, cons(4, [])), C109)
match: append(cons(X119, A117), B116, cons(X119, C118))
Unifier:
X119 :1
A117 :cons(2, [])
B116 :cons(3, cons(4, []))
C109 :cons(X119, C118)


NEW GOALS:
append(cons(2, []), cons(3, cons(4, [])), C118)


Depth: 1
Solving goal: append(cons(2, []), cons(3, cons(4, [])), C118)
Depth: 1
goal: append(cons(2, []), cons(3, cons(4, [])), C118)
match: append(cons(X128, A126), B125, cons(X128, C127))
Unifier:
X119 :1
A117 :cons(2, [])
B116 :cons(3, cons(4, []))
C109 :cons(X119, C118)
X128 :2
A126 :[]
B125 :cons(3, cons(4, []))
C118 :cons(X128, C127)


NEW GOALS:
append([], cons(3, cons(4, [])), C127)


Depth: 2
Solving goal: append([], cons(3, cons(4, [])), C127)
Depth: 2
goal: append([], cons(3, cons(4, [])), C127)
match: append([], B133, B133)
Unifier:
X119 :1
A117 :cons(2, [])
B116 :cons(3, cons(4, []))
C109 :cons(X119, C118)
X128 :2
A126 :[]
B125 :cons(3, cons(4, []))
C118 :cons(X128, C127)
B133 :cons(3, cons(4, []))
C127 :B133


All Goals Solved!!


MGU:
X119 : 1
A117 : cons(2, [])
B116 : cons(3, cons(4, []))
C109 : cons(X119, C118)
X128 : 2
A126 : []
B125 : cons(3, cons(4, []))
C118 : cons(X128, C127)
B133 : cons(3, cons(4, []))
C127 : B133


C = [1, 2, 3, 4]
User chose to continue
True
```

7. Now inside the "code" folder run the following command on the terminal:

`python3 query_solver.py -k ../output/sample_input.xml -q ../output/sample_query.xml`
for running the 'HC queries' in the HC file.

Terminal File Generated:

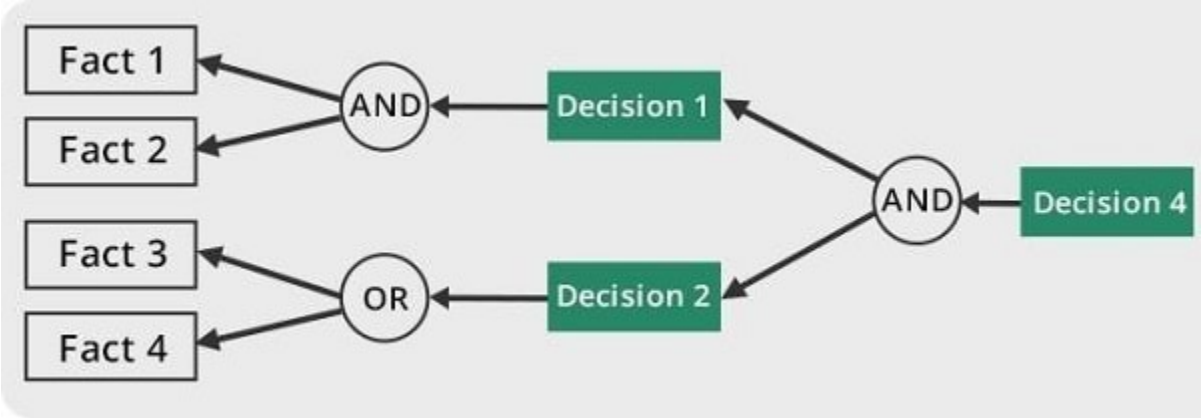


Trace File Generated:

```
FINISHED PARSING KB
FINISHED PARSING QUERY
Query:
sibling(reena, hitesh)


Depth: 0
Solving goal: sibling(reena, hitesh)
Depth: 0
goal: sibling(reena, hitesh)
match: sibling(X8, Y7)
Unifier:
X8 :reena
Y7 :hitesh


NEW GOALS:
sibling(reena, Z9)
sibling(Z9, hitesh)


Depth: 1
Solving goal: sibling(reena, Z9)
Depth: 1
goal: sibling(reena, Z9)
match: sibling(reena, suresh)
Unifier:
X8 :reena
Y7 :hitesh
Z9 :suresh


NEW GOALS:
sibling(suresh, hitesh)


Depth: 2
Solving goal: sibling(suresh, hitesh)
Depth: 2
goal: sibling(suresh, hitesh)
match: sibling(suresh, hitesh)
Unifier:
X8 :reena
Y7 :hitesh
Z9 :suresh


All Goals Solved!!


MGU:
X8 : reena
Y7 : hitesh
Z9 : suresh


True
```

3. About Backward Chaining:

Backward chaining is a concept in artificial intelligence that involves backtracking from the endpoint or goal to steps that lead to the endpoint. This type of chaining starts from the goal and moves backward to comprehend the steps that were taken to attain this goal. The backtracking process can also enable a person to establish logical steps that can be used to find other important solutions.



Backward chaining can be used in debugging, diagnostics, and prescription applications.

Properties of backward chaining:

1. The process uses an up-down approach (top to bottom).
2. It's a goal-driven method of reasoning.
3. The endpoint (goal) is subdivided into sub-goals to prove the truth of facts.
4. A backward chaining algorithm is employed in inference engines, game theories, and complex database systems.

5. The modus ponens inference rule is used as the basis for the backward chaining process. This rule states that if both the conditional statement (p->q) and the antecedent (p) are true, then we can infer the subsequent (q).

Advantages:

The result is already known, which makes it easy to deduce inferences. It's a quicker method of reasoning than forward chaining because the endpoint is available. In this type of chaining, correct solutions can be derived effectively if predetermined rules are met by the inference engine.

Disadvantages:

The process of reasoning can only start if the endpoint is known. It doesn't deduce multiple solutions or answers.It only derives data that is needed, which makes it less flexible than forward chaining.

4. Conclusion:

Backward chaining is an important method of reasoning in artificial intelligence. Backward chaining is important to developers that are interested in using goal-driven algorithms to design effective solutions in complex database systems.

Resource: *"backward chaining in artificial intelligence"*
https://www.section.io/engineering-education/forward-and-backward-chaining-in-ai/