"Heaven's Light is Our Guide"

Rajshahi University of Engineering & Technology
Department of Computer Science & Engineering

Lab Report

Course Code: CSE 3208

Course Title: Artificial Intelligence Sessional

Submitted By-	Submitted To-
Name: Sajidur Rahman Tarafder	**Md. Nasif Osman Khansur**
Department: CSE	**Lecturer**
Roll No: 2003154	**Department of CSE, RUET**
Section: C	
Session: 2020-21	

Experiment Title: Implementation of Forward Chaining and Backward Chaining using functions and OOP approach.

Introduction:

Artificial Intelligence (AI) uses reasoning techniques to derive conclusions from known facts and rules, enabling systems to make logical decisions. Two widely used reasoning methods in rule-based systems to derive conclusions, making decisions, are Forward Chaining and Backward Chaining.

Forward Chaining is a data-driven approach that starts with initial facts and applies rules to infer new facts until no more rules can be applied or a goal is reached. It is ideal for exploring all possible outcomes, such as in diagnostics.

Backward Chaining, in contrast, is goal-driven. It begins with a specific goal and works backward to determine if the goal can be achieved using the available facts and rules. This method is efficient for verifying hypotheses or answering specific queries.

Theory:

Forward Chaining:

Forward chaining is a data-driven approach that starts with a set of known facts and applies rules to infer new facts. The process continues until no new facts can be derived or a specific goal is reached. This method is exhaustive and ensures that all possible conclusions are derived from the given facts and rules. It is commonly used in systems where comprehensive reasoning is required, such as expert systems and diagnostic tools.

Algorithm:

1. Initialize a fact base with known facts.
2. Iterate through all rules:
   - If the rule's conditions are satisfied by the current facts, infer new facts.
   - Add the inferred facts to the fact base.
3. Repeat until no new facts can be inferred or the goal is reached.

Forward Chaining Code (Using Function):

Forward_Chaining_Using_Function.py

```python
facts = ["fever","cough"]

def rule1():
    if "fever" in facts and "infection" not in facts:
        facts.append("infection")
def rule2():
    if "cough" in facts and "infection" in facts  and "flu" not in facts:
        facts.append("flu")
def rule3():
    if "rash" in facts  and "infection" in facts and "measles" not in facts:
        facts.append("measles")


def main():
    rule1()
    rule2()
    rule3()
    print('Facts:',facts)
    print('Decision:',facts[-1])

main()
```

Terminal Output:

```
~/Desktop/CSE_3208 — -zsh
sajid@Sajids-MacBook-Air CSE_3208 % python3 Forward_Chaining_Using_Function.py
Facts: ['fever', 'cough', 'infection', 'flu']
Decision: flu
sajid@Sajids-MacBook-Air CSE_3208 %
```

Forward Chaining Code (OOP):

Forward_Chaining_OOP.py

```python
class Forward_Chaining:
    def __init__(self):
        self.facts = ["fever", "cough"]

    def rule1(self):
        if "fever" in self.facts and "infection" not in self.facts:
            self.facts.append("infection")

    def rule2(self):
        if "cough" in self.facts and "infection" in self.facts and "flu" not in self.facts:
            self.facts.append("flu")

    def rule3(self):
        if "rash" in self.facts and "infection" in self.facts and "measles" not in self.facts:
            self.facts.append("measles")

def main():
    Obj = Forward_Chaining()
    Obj.rule1()
    Obj.rule2()
    Obj.rule3()
    print('Facts:', Obj.facts)
    print('Decision:', Obj.facts[-1])

main()
```

Terminal Output:

```
~/Desktop/CSE_3208 — -zsh
sajid@Sajids-MacBook-Air CSE_3208 % python3 Forward_Chaining_OOP.py
Facts: ['fever', 'cough', 'infection', 'flu']
Decision: flu
sajid@Sajids-MacBook-Air CSE_3208 %
```

Backward Chaining:

Backward chaining is a goal-driven approach. It begins with a specific goal and works backward to determine if the goal can be satisfied using the given facts and rules. This method is efficient for verifying specific hypotheses, as it focuses only on the conditions necessary to achieve the goal. Backward chaining is widely used in applications like theorem proving and decision-making systems, where the objective is to validate a particular conclusion rather than derive all possible outcomes.

Algorithm:

1. Start with a goal.
2. Check if the goal is already in the known facts:
   - If yes, return success.
3. Otherwise, find rules that can infer the goal:
   - Recursively check if the rule's conditions can be satisfied.
4. If all conditions are satisfied, add the goal to the facts and return success.
5. If no rules can infer the goal, return failure.

Backward Chaining Code (Using Function):

Backward_Chaining_Using_Functions.py

```python
facts = ["fever", "cough"]

def rule1():
    if "fever" in facts and "infection" not in facts:
        facts.append("infection")
        return True
    return False

def rule2():
    if "cough" in facts and "infection" in facts and "flu" not in facts:
        facts.append("flu")
        return True
    return False

def rule3():
    if "rash" in facts and "infection" in facts and "measles" not in facts:
        facts.append("measles")
        return True
    return False

def backward_chaining(goal):
    if goal in facts:
        return True
    if goal == "infection":
        return rule1()
    elif goal == "flu":
        if backward_chaining("infection"):
            return rule2()
    elif goal == "measles":
        if backward_chaining("infection") and "rash" in facts:
            return rule3()
    return False

def main():
    goal = "flu"
    if backward_chaining(goal):
        print('Facts:', facts)
        print('Goal achieved.\nDecision:', goal)
    else:
        print('Facts:', facts)
        print('Goal not achieved')

main()
```

Terminal Output:

```
~/Desktop/CSE_3208 — -zsh
sajid@Sajids-MacBook-Air CSE_3208 % python3 Backward_Chaining_Using_Functions.py
Facts: ['fever', 'cough', 'infection', 'flu']
Goal achieved.
Decision: flu
sajid@Sajids-MacBook-Air CSE_3208 %
```

Backward Chaining Code (OOP):

Backward_Chaining_OOP.py

```python
class BackwardChaining:
    def __init__(self):
        self.facts = ["fever", "cough"]

    def rule1(self):
        if "fever" in self.facts and "infection" not in self.facts:
            self.facts.append("infection")
            return True
        return False

    def rule2(self):
        if "cough" in self.facts and "infection" in self.facts and "flu" not in self.facts:
            self.facts.append("flu")
            return True
        return False

    def rule3(self):
        if "rash" in self.facts and "infection" in self.facts and "measles" not in self.facts:
            self.facts.append("measles")
            return True
        return False

    def backward_chaining(self, goal):
        if goal in self.facts:
            return True
        if goal == "infection":
            return self.rule1()
        elif goal == "flu":
            if self.backward_chaining("infection"):
                return self.rule2()
        elif goal == "measles":
            if self.backward_chaining("infection") and "rash" in self.facts:
                return self.rule3()
        return False
def main():
    bc = BackwardChaining()
    goal = "flu"
    if bc.backward_chaining(goal):
        print('Facts:', bc.facts)
        print('Goal achieved.\nDecision:', goal)
    else:
        print('Facts:', bc.facts)
        print('Goal not achieved')

main()
```

Terminal Output:

```
~/Desktop/CSE_3208 — -zsh
sajid@Sajids-MacBook-Air CSE_3208 % python3 Backward_Chaining_OOP.py
Facts: ['fever', 'cough', 'infection', 'flu']
Goal achieved.
Decision: flu
sajid@Sajids-MacBook-Air CSE_3208 %
```

Discussion and Conclusion:

Forward and backward chaining are essential reasoning techniques in artificial intelligence, each suited for different scenarios. Forward chaining is a data-driven approach that derives all possible conclusions from a set of known facts by sequentially applying rules. It is exhaustive and ideal for comprehensive reasoning but may infer unnecessary facts unrelated to a specific goal. In contrast, backward chaining is a goal-driven approach that focuses on verifying whether a specific goal can be achieved by recursively checking the conditions of relevant rules. This makes it efficient for hypothesis testing, as it avoids unnecessary computations by exploring only the paths directly related to the goal. While forward chaining is better for deriving all possible outcomes, backward chaining is more efficient for validating specific conclusions. Both methods are fundamental in building intelligent systems capable of reasoning and decision-making, with the choice between them depending on the application's requirements.