Expert Systems Architecture and Design

---

1. Introduction to Expert Systems

Definition:

An **Expert System** is an artificial intelligence (AI) program that simulates the decision-making ability of a human expert. These systems are designed to solve complex problems within a specific domain using expert knowledge and reasoning techniques. They aim to replace or assist human experts in tasks like diagnosis, decision-making, and problem-solving.

Core Components of an Expert System:

1. **Knowledge Base:**
   This component contains domain-specific knowledge. The knowledge base stores facts and rules gathered from human experts or documents.
2. **Inference Engine:**
   The inference engine processes the information in the knowledge base. It applies logical rules to deduce new facts or solutions from existing ones. The engine can either:
   - Use **forward chaining** to start with known facts and work towards conclusions, or
   - Use **backward chaining** to begin with a goal and work backwards to determine the necessary conditions to achieve it.
3. **User Interface:**
   This is the point of interaction between the user and the system. The interface allows users to input queries and receive results from the system. An intuitive and user-friendly interface is crucial for system usability.
4. **Explanation Module:**
   This optional but valuable component explains the reasoning behind the system's decisions. It helps users understand the logical path the system followed to reach a conclusion, which enhances trust and system transparency.

Real-World Applications of Expert Systems:

- **Medical Diagnosis:** Systems like MYCIN were used to diagnose bacterial infections and suggest treatments.

- **Financial Planning:** Expert systems can assist in advising on loans, mortgages, and investments.
- **Industrial Process Control:** They monitor and manage complex industrial systems, such as power plants or manufacturing lines.

---

2. Rule-Based Expert Systems

Definition:

A **Rule-Based Expert System** uses predefined rules (typically in the form of "if-then" statements) to make decisions or draw conclusions. These systems apply logical deductions based on the rules in their knowledge base.

Structure of Rule-Based Systems:

- **Rules:** Each rule has two parts:
    - **IF** part (antecedent): Defines the condition to be checked.
    - **THEN** part (consequent): Specifies the action or conclusion when the condition is satisfied.

Example:
IF a patient has a fever AND a sore throat, THEN the system concludes it could be strep throat.

Inference Mechanism:

- **Forward Chaining:**
  In forward chaining, the system begins with known facts and applies rules to infer new facts until it reaches a conclusion. This method is data-driven, meaning it is triggered by the available data.
  Example: Starting from symptoms, forward chaining deduces potential diseases.
- **Backward Chaining:**
  Backward chaining works in the opposite direction. It starts with a goal (or hypothesis) and works backward to find evidence or conditions that support that goal. This method is goal-driven, focusing on proving or disproving a hypothesis.
  Example: To diagnose a disease, backward chaining tests whether symptoms match the hypothesized illness.

Advantages and Limitations:

- **Advantages:**
    - Straightforward implementation and easy to modify rules.
    - Transparent and easy to explain the reasoning process to users.
- **Limitations:**
    - Limited flexibility: New rules must be manually added, and the system may not adapt well to novel situations.
    - Poor scalability: As the number of rules grows, maintaining the system becomes more complex.
    - No learning capability: Cannot improve based on past performance unless rules are manually updated.

---

3. Case-Based Reasoning (CBR)

Definition:

Case-Based Reasoning (CBR) is an approach to problem-solving that uses past experiences or cases to solve new problems. Unlike rule-based systems, CBR does not require predefined rules. Instead, it relies on the idea that similar problems have similar solutions.

The CBR Process:

- **Retrieve:** The system identifies past cases that are similar to the current problem.
- **Reuse:** The solution from the most similar past case is adapted to the current situation.
- **Revise:** The system tests the adapted solution and, if necessary, makes corrections.
- **Retain:** The successful solution is stored as a new case in the case base for future reference.

Example of CBR:
In medical diagnosis, if a patient's symptoms closely resemble those of a previous patient, the system retrieves the case of the previous patient, reuses the solution (diagnosis and treatment), and revises it based on any differences in symptoms.

Advantages and Limitations:

- **Advantages:**
    - **Learning Capability:** The system improves over time by learning from new cases.
    - **Flexibility:** It can adapt solutions to fit new problems without requiring an exhaustive set of predefined rules.
- **Limitations:**
    - **Dependency on Case Base:** The system's effectiveness depends heavily on having a large and diverse case base.
    - **Complexity in Case Retrieval:** As the number of cases grows, retrieving the most relevant one can become time-consuming and computationally expensive.

---

4. Knowledge Engineering and Acquisition

Definition:

Knowledge Engineering is the process of designing and building expert systems by capturing and formalizing the knowledge of domain experts into a usable format. This includes not only structuring the knowledge but also ensuring that the system can apply it effectively.

Techniques for Knowledge Acquisition:

1. **Interviews:** Knowledge engineers interview domain experts to extract their problem-solving strategies.
2. **Observation:** Engineers observe domain experts as they perform tasks to identify implicit knowledge and decision-making patterns.
3. **Manual Extraction from Documents:** Knowledge can also be mined from documents, logs, or databases.
4. **Self-Learning Systems:** In some modern systems, the expert system can acquire new knowledge automatically by analyzing data (though this blurs the line between expert systems and machine learning systems).

Challenges in Knowledge Acquisition:

- **Tacit Knowledge:** Much of an expert's knowledge is implicit and not easily articulated.
- **Knowledge Volatility:** The expert's knowledge may change over time, requiring the system to be updated regularly.
- **Knowledge Bottlenecks:** The process of acquiring knowledge from experts can be time-consuming, especially if the experts are unavailable.

Knowledge Representation:
Knowledge must be structured into a form the system can process, often using:

- **Rules** (for rule-based systems),
- **Cases** (for case-based systems), or
- **Frames, Semantic Networks, or Ontologies** for more complex representations.

---

5. Building and Evaluating Expert Systems

Building Process:

1. **Identify the Problem Domain:**
   The first step is to select a narrow, well-defined domain of expertise where human experts are available.
2. **Design the Knowledge Base:**
   Gather and structure the knowledge from domain experts using interviews, document analysis, or observation.
3. **Implement the Inference Engine:**
   Decide whether the system will use forward chaining, backward chaining, case-based reasoning, or a hybrid approach.
4. **Design the User Interface:**
   Ensure the interface is simple enough for end-users to interact with easily, providing input and receiving feedback.
5. **Explanation Mechanism:**
   If required, include an explanation module to provide insights into the system's decision-making process.

Evaluation Process:

- **Verification:**
  Ensure that the system's logic and reasoning are accurate. Verification checks whether the system behaves as intended.
- **Validation:**
  Validate that the system's outputs are correct and align with real-world expert solutions. Validation ensures the system performs as well as or better than human experts.
- **Performance Evaluation:**
  Measure the system's efficiency, accuracy, and scalability. Performance evaluations assess whether the system can handle a variety of cases in a timely and effective manner.

Key Metrics for Evaluation:

1. **Accuracy:** How often the system provides correct solutions or conclusions.
2. **Efficiency:** The system's ability to provide results quickly, especially under heavy workloads.
3. **User Satisfaction:** The ease of use and trust placed in the system by end-users.
4. **Reliability:** The system's ability to perform consistently over time.

Practical Example:
For an expert system in the medical field, evaluate the system by comparing its diagnostic accuracy with real doctors' decisions over time.

Summary

Expert systems are powerful tools in fields requiring specialized knowledge, but they come with challenges in design, implementation, and maintenance. Understanding the architecture, techniques for knowledge acquisition, and methods for evaluating the systems are crucial for their successful development. Through rule-based or case-based systems, and with proper engineering practices, expert systems can significantly enhance decision-making and problem-solving in specialized domains.

Module: Architecture and Design of Expert Systems

Duration: 3 Hours
Mode: Lecture and Discussion with Interactive Examples

Module Overview

This module introduces students to the architecture of expert systems, focusing on their core components and functionality. Students will explore rule-based systems, case-based reasoning, and the methodologies used in knowledge engineering and acquisition. The session concludes with practical insights into building and evaluating expert systems.

Learning Objectives:

By the end of the module, students will be able to:

- Understand the architecture of expert systems and their components.
- Differentiate between rule-based and case-based expert systems.
- Comprehend the process of knowledge engineering and knowledge acquisition.
- Identify best practices in building and evaluating expert systems.

Module Outline:

---

1. Introduction to Expert Systems (20 minutes)

- **Definition:**
  Expert systems are AI programs that emulate the decision-making ability of a human expert.
- **Applications:**
  - Medical diagnosis
  - Financial decision-making
  - Industrial process control
- **Key Components:**

- **Knowledge Base:** Contains domain-specific facts and heuristics.
- **Inference Engine:** Applies rules to the knowledge base to deduce new information.
- **User Interface:** Allows interaction between the user and the system.
- **Explanation Module:** Justifies the decisions made by the system.

Interactive Discussion:

- How do expert systems compare to human decision-making?
- Real-world examples of expert systems.

---

2. Rule-Based Expert Systems (30 minutes)

- **Definition:**
  Rule-based systems use a set of "if-then" rules to make decisions or solve problems.
- **Key Features:**
  - Simple rule structure.
  - High transparency in the decision-making process.
  - Easier to understand and implement in limited domains.
- **Forward vs. Backward Chaining:**
  - **Forward Chaining:** Starts from known facts and applies rules to arrive at conclusions.
  - **Backward Chaining:** Starts with goals or hypotheses and works backward to find supporting evidence.
- **Advantages and Limitations:**
  - **Advantages:** Well-structured, easy to modify.
  - **Limitations:** Rigid and often struggles with complex, dynamic environments.

Activity:

- Students are asked to write simple "if-then" rules for diagnosing car engine problems using a rule-based system.

---

3. Case-Based Reasoning (CBR) (30 minutes)

- **Definition:**
  Case-based reasoning solves new problems by adapting solutions that were used to solve previous, similar problems.
- **CBR Cycle:**
  - **Retrieve:** Identify past cases similar to the current problem.
  - **Reuse:** Adapt the solution of the retrieved case to the new problem.
  - **Revise:** Test and adapt the proposed solution.
  - **Retain:** Store the solution for future reference.
- **Applications:**
  - Legal case solving, helpdesk systems, and medical diagnosis.
- **Advantages and Limitations:**
  - **Advantages:** Learns from past experience, flexible problem-solving.
  - **Limitations:** Requires a large, well-maintained case database.

Class Discussion:

- Discuss practical examples where CBR might outperform rule-based systems.

---

4. Knowledge Engineering and Acquisition (25 minutes)

- **Definition:**
  Knowledge engineering is the process of designing and building expert systems by acquiring knowledge from experts and structuring it into a usable format.
- **Knowledge Acquisition Techniques:**
  - **Interviews:** Consulting domain experts.
  - **Observation:** Watching domain experts in action.
  - **Manual Extraction:** Mining documents, logs, or databases.
- **Challenges in Knowledge Acquisition:**

    - Knowledge is often tacit and hard to articulate.
    - Domain experts may have difficulty explaining complex processes clearly.

Activity:

- Engage students in a role-play where one group acts as domain experts, and the other as knowledge engineers to extract knowledge on a specific domain, e.g., diagnosing an illness.

---

5. Building and Evaluating Expert Systems (30 minutes)

- **Building Process:**
    - **Identify the Problem Domain:** Choose a well-defined, narrow area of expertise.
    - **Construct the Knowledge Base:** Acquire and structure relevant knowledge.
    - **Implement the Inference Engine:** Define how the system will draw conclusions.
    - **Design User Interfaces:** Ensure usability for both experts and non-experts.
- **Evaluation Techniques:**
    - **Verification:** Ensure the system's rules and logic are correct.
    - **Validation:** Check if the system provides accurate and reliable results.
    - **Performance Evaluation:** Measure how efficiently the system solves problems.
- **Common Metrics for Evaluation:**
    - Accuracy, response time, user satisfaction, and reliability.

Example Discussion:

- Analyze an existing expert system and discuss how it might be improved in terms of evaluation metrics.

Conclusion and Review (15 minutes)

- Recap the key points of expert system architecture, rule-based systems, case-based reasoning, and the knowledge engineering process.
- Q&A session to clarify any doubts or deeper exploration into specific concepts.

Assignments and Further Reading:

- **Assignment:** Develop a basic rule-based expert system using pseudocode for a simple problem (e.g., determining loan eligibility).
- **Reading Material:**
    - "Building Expert Systems" by Frederick Hayes-Roth.
    - "Expert Systems: Principles and Programming" by Joseph Giarratano.

1. Rule-Based Expert Systems

We'll use Python's **pyknow** library, which is great for building simple rule-based systems. Let's create a basic rule-based expert system for medical diagnosis.

Example: Medical Diagnosis (Rule-Based System)

```
# Install pyknow library first:
# pip install pyknow
from pyknow import *
class Patient(Fact):
  """Info about the patient"""
  pass
class MedicalDiagnosis(KnowledgeEngine):
  @Rule(Patient(fever=True, sore_throat=True))
  def strep_throat(self):
    print("Diagnosis: Strep Throat")
  @Rule(Patient(cough=True, runny_nose=True))
  def common_cold(self):
    print("Diagnosis: Common Cold")
# Creating an instance of the knowledge engine
engine = MedicalDiagnosis()
# Resetting the engine to prepare it
engine.reset()
# Declaring patient facts (forward chaining example)
engine.declare(Patient(fever=True, sore_throat=True))
# Running the inference engine
engine.run()
```

Explanation:

- The system will diagnose the patient based on the provided symptoms.
- You can declare different symptoms, and the system will print the diagnosis.

---

2. Case-Based Reasoning (CBR)

Here's an implementation of a simplified CBR in Python. We will use a case of a patient diagnosis, where the system retrieves the most similar case from its memory.

Example: Medical Diagnosis (Case-Based Reasoning)

```
class Case:
  def __init__(self, symptoms, diagnosis):
    self.symptoms = symptoms
```

```
        self.diagnosis = diagnosis

class CaseBasedReasoningSystem:
    def __init__(self):
        self.case_base = []

    def add_case(self, case):
        self.case_base.append(case)

    def retrieve(self, new_symptoms):
        # Simple similarity measure: counts how many symptoms match
        best_match = None
        highest_similarity = 0

        for case in self.case_base:
            similarity = sum(1 for symptom in case.symptoms if symptom in new_symptoms)

            if similarity > highest_similarity:
                highest_similarity = similarity
                best_match = case

        return best_match

# Defining past cases
case1 = Case(symptoms=["fever", "sore throat"], diagnosis="Strep Throat")
case2 = Case(symptoms=["cough", "runny nose"], diagnosis="Common Cold")

# Creating the system and adding cases to the case base
cbr_system = CaseBasedReasoningSystem()
cbr_system.add_case(case1)
cbr_system.add_case(case2)

# New patient's symptoms
new_patient_symptoms = ["fever", "sore throat"]

# Retrieving the most similar case
matched_case = cbr_system.retrieve(new_patient_symptoms)
if matched_case:
    print(f"Diagnosis based on similar case: {matched_case.diagnosis}")
else:
    print("No similar case found.")
```

Explanation:

- The system stores previous cases and retrieves the most similar one based on symptoms.

---

3. Knowledge Engineering and Acquisition (Rule-Based Example)

For knowledge acquisition, let's use a simplified system for rule-based knowledge representation using Python dictionaries.

Example: Simplified Rule-Based Knowledge System

```
class KnowledgeBase:
  def __init__(self):
    self.rules = {
      "fever and sore throat": "Strep Throat",
      "cough and runny nose": "Common Cold"
    }

  def infer(self, symptoms):
    condition = " and ".join(symptoms)
    if condition in self.rules:
      return self.rules[condition]
    else:
      return "No diagnosis available for given symptoms."

# Creating a knowledge base
knowledge_base = KnowledgeBase()

# Symptoms entered by the user
patient_symptoms = ["fever", "sore throat"]

# Getting the diagnosis
diagnosis = knowledge_base.infer(patient_symptoms)
print(f"Diagnosis: {diagnosis}")
```

Explanation:

- The system uses a simple dictionary to store rules, and it matches user symptoms to those rules to provide a diagnosis.

---

4. Building and Evaluating Expert Systems

Let's build and evaluate an expert system for a small medical diagnosis using rules and facts in Python.

Example: Full Rule-Based Expert System with Evaluation

```
from pyknow import *

class Symptom(Fact):
  """Symptom information"""
  pass

class MedicalExpertSystem(KnowledgeEngine):
  @Rule(Symptom(fever=True, cough=True))
  def diagnose_flu(self):
    print("Diagnosis: Flu")

  @Rule(Symptom(sore_throat=True, fever=True))
  def diagnose_strep_throat(self):
    print("Diagnosis: Strep Throat")

  @Rule(Symptom(runny_nose=True, cough=True))
  def diagnose_common_cold(self):
    print("Diagnosis: Common Cold")

# Creating and initializing the expert system
expert_system = MedicalExpertSystem()
expert_system.reset()

# Adding facts (symptoms)
expert_system.declare(Symptom(fever=True, cough=True))
expert_system.declare(Symptom(sore_throat=True, fever=True))

# Running the expert system inference
expert_system.run()
```

Explanation:

- This example shows a basic medical diagnosis system, which evaluates multiple conditions based on provided symptoms and generates appropriate diagnoses.

Conclusion

These Python codes demonstrate basic implementations of rule-based expert systems and case-based reasoning systems. In more advanced setups, you could integrate databases for storing and retrieving large-scale cases or rules, and use machine learning techniques for knowledge acquisition or case retrieval.

Paper and Pen Simulation

1. Rule-Based Expert System for Medical Diagnosis

Simulation Steps

Problem: A patient is showing symptoms of *fever* and *sore throat*. You want to diagnose their condition using predefined rules.

Rules:

1. If the patient has *fever* and *sore throat*, then the diagnosis is *Strep Throat*.
2. If the patient has *cough* and *runny nose*, then the diagnosis is *Common Cold*.

Steps:

1. **Input:** The patient has a *fever* and a *sore throat*.
2. **Check Rule 1:** Does the patient have both *fever* and *sore throat*?
    - Yes, the patient meets this condition.
3. **Diagnosis:** The system concludes that the diagnosis is *Strep Throat*.
4. **Output:** Print "Diagnosis: Strep Throat."

Simulation:

- Given Symptoms: *fever*, *sore throat*
- Rule Applied: If *fever* and *sore throat*, then *Strep Throat*.
- Result: Diagnosis is *Strep Throat*.

---

2. Case-Based Reasoning for Medical Diagnosis

Simulation Steps

Problem: A patient presents with symptoms of *fever* and *sore throat*. You want to diagnose their condition by comparing the symptoms to previously encountered cases.

Cases in Memory:

Case 1: Symptoms: *fever*, *sore throat*, Diagnosis: *Strep Throat*
Case 2: Symptoms: *cough*, *runny nose*, Diagnosis: *Common Cold*

Steps:

1. **Input:** The patient has *fever* and *sore throat*.

2. **Similarity Check:**
    - Compare the patient's symptoms to Case 1: How many symptoms match? Both *fever* and *sore throat* match (2 matches).
    - Compare the patient's symptoms to Case 2: No symptoms match (0 matches).
3. **Best Match:** Case 1 has the highest similarity (2 matches).
4. **Diagnosis:** Based on Case 1, the diagnosis is *Strep Throat*.
5. **Output:** Print "Diagnosis based on similar case: Strep Throat."

Simulation:

1. Given Symptoms: *fever*, *sore throat*
2. Case Base:
3. Case 1: *fever*, *sore throat* → *Strep Throat* (2 matches)
4. Case 2: *cough*, *runny nose* → *Common Cold* (0 matches)
5. Best Match: Case 1 (2 matches)
6. Result: Diagnosis is *Strep Throat*.

---

3. Knowledge Engineering and Acquisition using Rule-Based System

Simulation Steps

Problem: A patient presents with *fever* and *sore throat*, and you need to diagnose their condition based on the knowledge rules stored in the system.

Knowledge Base:

Rule 1: fever and sore throat → Strep Throat
Rule 2: cough and runny nose → Common Cold

Steps:

1. **Input:** The patient has *fever* and *sore throat*.
2. **Check Rule 1:** Do the symptoms match the condition in Rule 1 (i.e., *fever* and *sore throat*)?
3. Yes, both symptoms match.
4. **Diagnosis:** The rule concludes the diagnosis is *Strep Throat*.
5. **Output:** Print "Diagnosis: Strep Throat."

Simulation:

- Given Symptoms: *fever*, *sore throat*
- Rule 1: *fever* and *sore throat* → *Strep Throat*
- Match Found: Yes
- Result: Diagnosis is *Strep Throat*.

---

4. Building and Evaluating Expert Systems

Simulation Steps

Problem: A patient presents with various symptoms. The expert system uses predefined rules to diagnose the patient's condition. The patient has *fever*, *cough*, and *sore throat*.

Rules:

1. If the patient has *fever* and *cough*, the diagnosis is *Flu*.
2. If the patient has *sore throat* and *fever*, the diagnosis is *Strep Throat*.
3. If the patient has *cough* and *runny nose*, the diagnosis is *Common Cold*.

Steps:

1. **Input:** The patient has *fever*, *cough*, and *sore throat*.
2. **Rule 1 Check:** Does the patient have both *fever* and *cough*?
3. Yes, the patient meets this condition.
4. **Diagnosis 1:** The system concludes that the diagnosis is *Flu*.
5. **Rule 2 Check:** Does the patient have both *fever* and *sore throat*?
6. Yes, the patient meets this condition as well.
7. **Diagnosis 2:** The system also concludes that the diagnosis could be *Strep Throat*.
8. **Multiple Diagnoses:** The system suggests two possible diagnoses: *Flu* and *Strep Throat*.
9. **Output:** Print "Possible Diagnoses: Flu, Strep Throat."

Simulation:

- Given Symptoms: fever, cough, sore throat
- Rule 1: fever and cough → Flu (Match)
- Rule 2: sore throat and fever → Strep Throat (Match)
- Result: Possible Diagnoses are Flu and Strep Throat.

Summary of the Simulations

Each simulation works by walking through the symptoms or facts presented to the expert system and matching them to predefined rules or cases in memory. These steps help visualize how the system processes input and reaches a diagnosis or solution.

1. Rule-Based Expert Systems for Medical Diagnosis (No Computation)

Since rule-based systems typically rely on logic rather than computation, there isn't much numerical calculation here. The rules either match or don't match, based on the presence or absence of symptoms.

We can simulate how a system might compute **confidence values** for multiple diagnoses by assigning weights to symptoms. Here's an enhanced pen-and-paper simulation with confidence levels:

Problem: A patient shows the symptoms of *fever* and *sore throat*.

Rules and Confidence Levels:

1. If the patient has *fever* (confidence = 0.7) and *sore throat* (confidence = 0.6), then the diagnosis is *Strep Throat* (combined confidence = 0.7 × 0.6 = **0.42**).
2. If the patient has *cough* (confidence = 0.5) and *runny nose* (confidence = 0.4), then the diagnosis is *Common Cold*.

Steps:

1. **Input:** The patient has *fever* and *sore throat*.
2. **Calculate Confidence for Strep Throat:** Multiply the confidence of *fever* (0.7) by the confidence of *sore throat* (0.6): 0.7×0.6=0.42
3. **Compare Confidence Levels:** For *Common Cold*, no symptoms match, so confidence remains 0.
4. **Conclusion:** The highest confidence is 0.42 for *Strep Throat*.
5. **Output:** Print "Diagnosis: Strep Throat with confidence level of 0.42."

Simulation (with Computation):

- Given Symptoms: *fever* (0.7), *sore throat* (0.6)
- Computed Confidence for *Strep Throat*: 0.42
- No match for *Common Cold* (confidence = 0)
- Result: Diagnosis is *Strep Throat* with confidence 0.42.

---

2. Case-Based Reasoning (CBR) with Similarity Computation

In CBR, we can compute the similarity between cases based on the number of matching symptoms. Let’s simulate that with a simple score-based approach.

Simulation (with Similarity Computation)

Problem: A patient shows the symptoms *fever* and *sore throat*.

Cases in Memory (with Symptom Matching Scores):

1. Case 1: *fever*, *sore throat* (score = 2)
2. Case 2: *cough*, *runny nose* (score = 0)

Steps:

1. **Input:** The patient has *fever* and *sore throat*.
2. **Score Calculation for Case 1:**
3. The patient has *fever* (1 match) and *sore throat* (1 match).
4. Total score for Case 1: 1 + 1 = **2**.
5. **Score Calculation for Case 2:**
6. No symptoms match in Case 2.
7. Total score for Case 2: 0 + 0 = **0**.
8. **Best Match:** Case 1 has the highest score (2 matches).
9. **Conclusion:** Based on Case 1, the diagnosis is *Strep Throat*.
10. **Output:** Print "Diagnosis based on similar case: Strep Throat."

Simulation (with Computation):

- Given Symptoms: *fever*, *sore throat*
- Score for Case 1: 2
- Score for Case 2: 0
- Best Match: Case 1
- Result: Diagnosis is *Strep Throat*.

---

3. Knowledge Engineering: Computation of Confidence Levels

In a knowledge-based system, we can calculate the **weighted match score** of symptoms against each rule.

Simulation (with Symptom Weighting)

Problem: A patient presents *fever* and *sore throat*.

Knowledge Base:

1. Rule 1: *fever* (weight = 0.7) and *sore throat* (weight = 0.6) → *Strep Throat*.
2. Rule 2: *cough* (weight = 0.5) and *runny nose* (weight = 0.4) → *Common Cold*.

Steps:

1. **Input:** The patient has *fever* and *sore throat*.
2. **Weight Calculation for Strep Throat:** Multiply the weights of the matching symptoms: 0.7×0.6=0.42
3. **Weight Calculation for Common Cold:** No matching symptoms, so the score is 0.
4. **Conclusion:** The rule for *Strep Throat* gives a confidence of 0.42.
5. **Output:** Print "Diagnosis: Strep Throat with confidence level of 0.42."

Simulation (with Computation):

- Given Symptoms: *fever*, *sore throat*
- Weight for *Strep Throat*: 0.42
- Weight for *Common Cold*: 0
- Result: Diagnosis is *Strep Throat* with confidence 0.42.

---

4. Building and Evaluating Expert Systems thru Multiple Rules and Confidence

Here we calculate confidence levels for multiple diagnoses and see how we evaluate them.

Pen-and-Paper Simulation (with Multiple Diagnoses and Confidence Levels)

Problem: A patient has *fever*, *cough*, and *sore throat*.

Rules:

1. If the patient has *fever* (weight = 0.7) and *cough* (weight = 0.5), then the diagnosis is *Flu*.
2. If the patient has *sore throat* (weight = 0.6) and *fever* (weight = 0.7), then the diagnosis is *Strep Throat*.
3. If the patient has *cough* (weight = 0.5) and *runny nose* (weight = 0.4), then the diagnosis is *Common Cold*.

Steps:

1. **Input:** The patient has *fever*, *cough*, and *sore throat*.
2. **Rule 1 (Flu) Calculation:** Multiply the weights of *fever* and *cough*: 0.7×0.5=0.35
3. Confidence for *Flu*: **0.35**.
4. **Rule 2 (Strep Throat) Calculation:** Multiply the weights of *fever* and *sore throat*: 0.7×0.6=0.42
5. Confidence for *Strep Throat*: **0.42**.
6. **Rule 3 (Common Cold) Calculation:** No matching *runny nose* symptom, so confidence remains 0.
7. **Conclusion:** The system suggests two possible diagnoses: *Flu* (0.35) and *Strep Throat* (0.42).
8. **Output:** Print "Possible Diagnoses: Flu (0.35), Strep Throat (0.42)."

Simulation (with Computation):

- Given Symptoms: fever, cough, sore throat
- Confidence for Flu: 0.35
- Confidence for Strep Throat: 0.42
- Confidence for Common Cold: 0
- Result: Possible Diagnoses are Flu (0.35) and Strep Throat (0.42).
-

Conclusion

In these enhanced pen-and-paper simulations, we have incorporated computations like:

- Confidence levels in rule-based systems
- Symptom similarity scores in case-based reasoning
- Weighting of symptoms in knowledge engineering

These computations give more depth to the logical flow and help simulate how systems make decisions with numerical evaluations.