USN | | | | | | | | | | |

Internal Assessment Test 2 – July 2024

Sub:	**Artificial Intelligence**					Sub Code:	BAI402	Branch:	AIML
Date:	**/7/2024**	Duration:	90 min	Max Marks:	50	Sem/Sec:	IV /A, B & C		OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	What do you mean by Knowledge Based Agent and also explain its operations and architecture in detail?	10	CO1	L1
2	What is proposition logic? Explain the proposition logic representation with types of inference rules?	10	CO2	L2
3	Differentiate between Unification & Resolution with examples?	10	CO3	L2
4	Anything any one eats is called food. Mita likes all kinds of food. Burger is a food. Mango is a food. John eats pizza. John eats everything Mita eats " **Construct** these sentences into formulae in predicate logic and then to program clauses. Use resolution algorithm to answer the following. i. What food does John eat? ii.Does Mita like pizza? iii.Which food does John Like? iv.Who likes what foods? v. Prove the statement "Mita likes pizza and burger" using resolution.	10	CO3	L3
5	Convert the following statement into its Equivalent Predicate Logic from i) Marcus was a man ii) Marcus was a Pompeian iii) All Pompeians were Romans iv) Caesar was a Ruler v) All Romans were either loyal to Caesar of hated him. vi) Everyone is loyal to someone vii) People only try to assassinate rulers they are not loyal to. viii) Marcus tried to assassinate Caesar.	10	CO3	L3
6	Illustrate Forward chaining & Backward Chaining in detail with example.	10	CO3	L3

Faculty Signature CCI Signature HOD Signature

----------All the Best----------

CO-PO and CO-PSO Mapping																			
Course Outcomes		**Blooms Level**	**Modules covered**	**PO1**	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**	**PSO3**	**PSO4**
CO1	Apply knowledge of agent architecture, searching and	**L1, L2**	**1**	2	2	1	-	-	-	-	-	-	-	-	-	1	-	1	-

	reasoning techniques for different applications.																		
CO2	Compare various Searching and Inferencing Techniques.	**L2, L3**	**2,3,4**	2	3	1	-	-	-	-	-	-	-	-	-	-	-	-	-
CO3	Develop knowledge base sentences using propositional logic and first order logic	**L2**	**3, 4**	3	2	1	-	-	-	-	-	-	-	-	-	-	-	-	-
CO4	Describe the concepts of quantifying uncertainty	**L2**	**5**	3	2	2	-	-	-	-	-	-	-	-	-	-	-	1	-
CO5	Use the concepts of Expert Systems to build applications.	**L2, L3**	**4,5**	3	2	3	-	1	-	-	-	-	-	-	-	1	-	1	-

SCHEME & SOLUTIONS

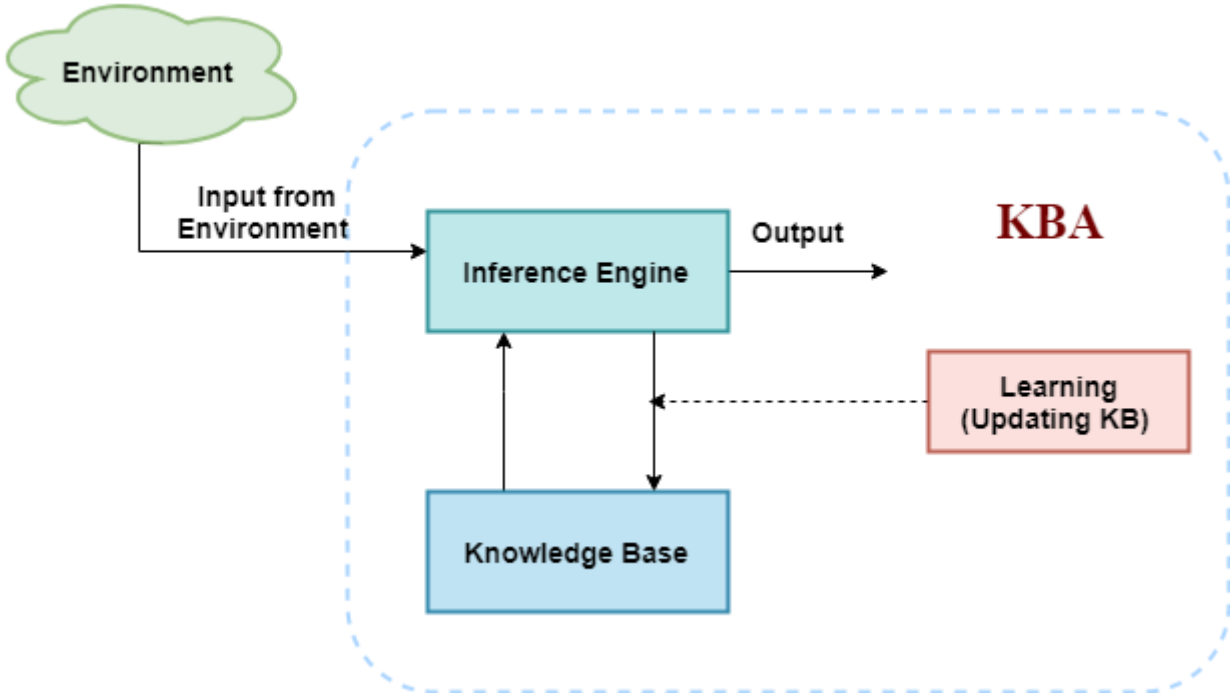
1) What do you mean by Knowledge Based Agent and also explain its operations and architecture in detail? 10 CO1

A Knowledge-Based Agent (KBA) is a type of intelligent agent that uses a knowledge base to make decisions and perform tasks. The knowledge base consists of information about the world, which the agent uses to reason, make inferences, and solve problems. Here is a detailed explanation of its operations and architecture:

Operations of a Knowledge-Based Agent

1. **Perception**: The agent perceives its environment through sensors, which gather data about the current state of the world.
2. **Knowledge Update**: The agent updates its knowledge base with the new information obtained from its sensors. This step often involves integrating new data with existing knowledge, resolving conflicts, and ensuring consistency.
3. **Inference**: The agent uses reasoning algorithms to infer new facts from the knowledge base. This process can involve logical deduction, probabilistic reasoning, or other forms of inference.
4. **Decision Making**: Based on the inferences made, the agent decides on a course of action. This step involves selecting an appropriate action that aligns with the agent's goals and the current state of the world.
5. **Action Execution**: The agent performs the selected action using its actuators, which may involve interacting with the environment or other agents.
6. **Learning**: Some knowledge-based agents also incorporate learning mechanisms to improve their knowledge base and reasoning capabilities over time. This can involve learning from past experiences, feedback from the environment, or new data.

Architecture of a Knowledge-Based Agent



The architecture of a Knowledge-Based Agent typically includes the following components:

1. **Sensors**: Devices or methods used to gather information from the environment. Sensors convert physical stimuli into data that the agent can process.
2. **Actuators**: Mechanisms through which the agent interacts with the environment. Actuators can be physical devices (like robotic arms) or software mechanisms (like sending messages or executing commands).
3. **Knowledge Base**: The central repository of knowledge for the agent. This includes:
    - **Facts**: Known truths about the world.
    - **Rules**: Logical statements that define relationships between facts and guide reasoning.
    - **Ontologies**: Structured frameworks that define the concepts and relationships within a domain.
4. **Inference Engine**: The component that performs reasoning tasks. It uses the rules in the knowledge base to infer new information and make decisions. Common inference techniques include:
    - **Deductive Reasoning**: Deriving specific conclusions from general rules and facts.
    - **Inductive Reasoning**: Inferring general rules from specific instances.
    - **Abductive Reasoning**: Inferring the best explanation for a set of observations.
5. **Knowledge Acquisition Component**: This component is responsible for updating the knowledge base with new information. It can involve manual entry, automated data collection, machine learning, and natural language processing.
6. **Planner**: Some agents include a planning component that generates sequences of actions to achieve specific goals. The planner uses the knowledge base and inference engine to develop and evaluate potential plans.
7. **Learning Component**: This optional component allows the agent to improve its performance over time. It can involve various learning algorithms, such as supervised learning, reinforcement learning, or unsupervised learning.

Example of a Knowledge-Based Agent in Action

Consider a medical diagnosis agent that assists doctors in diagnosing diseases:

1. **Perception**: The agent collects patient data, including symptoms, medical history, and test results.
2. **Knowledge Update**: The agent updates its knowledge base with the new patient data.

3. **Inference**: Using its rules and medical knowledge, the agent infers possible diagnoses and identifies the most likely ones.
4. **Decision Making**: The agent suggests a diagnosis and recommends further tests or treatments based on its inferences.
5. **Action Execution**: The agent communicates its diagnosis and recommendations to the doctor.
6. **Learning**: The agent updates its knowledge base with feedback from the doctor and patient

2.What is proposition logic? Explain the proposition logic representation with types of inference rules? 10 CO2

ANSWER:-

Propositional logic, also known as sentential logic or statement logic, is a branch of logic that deals with propositions (statements) that can be either true or false, but not both. It focuses on the relationships between these propositions and the rules that govern how to derive new propositions from existing ones.

Propositional Logic Representation

Here's how propositions and their relationships are represented in propositional logic:

1. **Propositions (p, q, r):** These are basic statements that are either true (T) or false (F). They represent atomic facts or simple ideas.
2. **Logical Connectives:** These are symbols that connect propositions and define the relationship between them. Common connectives include:
    - **Negation (¬):** NOT (flips the truth value)
    - **Conjunction (∧):** AND (both propositions must be true)
    - **Disjunction (∨):** OR (at least one proposition must be true)
    - **Implication (→):** IF...THEN (the first proposition implies the second)
    - **Equivalence (↔):** IF and ONLY IF (both propositions have the same truth value)
3. **Compound Propositions:** These are propositions formed by combining simpler propositions using logical connectives. Examples:
    - ~(p ∧ q) (NOT (p AND q))
    - (p → q) ∨ r ((p IMPLIES q) OR r)
4. **Truth Tables:** These are tables that show the truth value of a compound proposition for all possible combinations of truth values of its constituent propositions. They are essential for evaluating the logical behavior of connectives.

Types of Inference Rules

Inference rules are a set of techniques for deriving new propositions from existing ones that are known to be true. Here are some common inference rules in propositional logic:

1. **Modus Ponens:** If P implies Q, and P is true, then Q must be true. (P → Q, P ⊢ Q)
2. **Modus Tollens:** If P implies Q, and Q is false, then P must be false. (P → Q, ¬Q ⊢ ¬P)
3. **Hypothetical Syllogism:** If P implies Q, and Q implies R, then P implies R. (P → Q, Q → R ⊢ P → R)
4. **Disjunctive Syllogism:** If P or Q is true, and P is false, then Q must be true. (P ∨ Q, ¬P ⊢ Q)
5. **De Morgan's Laws:** These laws relate the negation of conjunctions and disjunctions.
    - ¬(P ∧ Q) is equivalent to (¬P ∨ ¬Q)
    - ¬(P ∨ Q) is equivalent to (¬P ∧ ¬Q)
6. **Double Negation:** ¬¬P is equivalent to P (removing double negation)

3. Differentiate between Unification & Resolution with examples? 10 CO3

Unification vs. Resolution in Logic

Both unification and resolution are fundamental techniques used in automated reasoning and theorem proving, but they serve distinct purposes within the process. Here's how they differ:

Unification:

- **Goal:** Finds a substitution for variables that makes two expressions identical (except for predicate symbols, functions, and constants).
- **Focus:** Identifies variable equivalences for potential resolution.
- **Example:**
  - Expression 1: Loves(x, Mary)
  - Expression 2: Loves(Bill, y)
  - Unification: Substitute x with Bill (ignoring predicates and constants)
  - Result: Both expressions become Loves(Bill, Mary) after unification.

Resolution:

- **Goal:** Derives new clauses (logical statements) from existing ones.
- **Focus:** Explores logical implications based on complementary literals.
- **Example:**
  - Clause 1: ¬Loves(x, John) ∨ Loves(x, Mary) (if not John, then Mary)
  - Clause 2 (negated goal): ¬Loves(Bill, John) (Bill doesn't love John)
  - Resolution: Combine remaining literals after unification (Loves(x, Mary) from clause 1, ¬Loves(Bill, John) from clause 2)
  - Result: New clause: Loves(Bill, Mary) (derived new statement)

Key Differences:

Feature	Unification	Resolution
Purpose	Finds variable substitutions for matching expressions	Derives new clauses through literal combination
Input	Two expressions (usually literals)	Existing clauses in a knowledge base
Output	Substitution for variables (if successful)	New clause (resolvent)
Role in reasoning	Enables variable matching for potential resolution	Explores logical implications between clauses

drive_spreadsheetExport to Sheets

Relationship:

Unification plays a crucial role within resolution. In the resolution process, unification is used to find variable substitutions that make two literals complementary (same predicate, opposite negation) before they can be combined to generate a new clause (resolvent).

4

Anything any one eats is called food. Mita likes all kinds of food. Burger is a food. Mango is a food. John eats pizza. John eats everything Mita eats "
Construct these sentences into formulae in predicate logic and then to program clauses
Use resolution algorithm to answer the following.
i. What food does John eat?
ii.Does Mita like pizza?
iii.Which food does John Like?
iv.Who likes what foods?
v. Prove the statement "Mita likes pizza and burger" using resolution.

(10) CO3

a. John likes all kind of food.

b. Apple and vegetable are food

c. Anything anyone eats and not killed is food.

d. Anil eats peanuts and still alive

e. Harry eats everything that Anil eats.

f. John likes peanuts.

a. ∀x: food(x) likes (John, x)

b. food(Apple) A food(vegetables)

c. Vx Vy: eats(x, y)^-killed(x) → food(y)

d. eats (Anil, Peanuts) A alive(Anil).

e. ∀x: eats(Anil, x) → eats (Harry, x)

f. vx:-killed(x) alive(x)added predicates

g. Vx: alive(x)→killed(x))h. likes (John, Peanuts)

Eliminate all implication (→) and rewrite

a. ∀x: food(x) →likes(John, x)

b. food(Apple) A food(vegetables)

c. ∀x Vy: eats(x, y) - killed(x) food(y)

d. eats (Anil, Peanuts) A alive(Anil).

e. ∀x: eats(Anil, x) → eats (Harry, x)

f. ∀x: killed(x) → alive(x)

g. Vx: alive(x) →- killed(x)

h. likes (John, Peanuts)

a. ∀x ¬ food(x) V likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)

d. eats (Anil, Peanuts) Λ alive(Anil)

e. ∀x ¬ eats(Anil, x) V eats(Harry, x)

f. ∀x¬ [¬ killed(x)] V alive(x)

g. ∀x ¬ alive(x) V ¬ killed(x)

h. likes(John, Peanuts).

- **Move negation (¬)inwards and rewrite**

a. ∀x ¬ food(x) V likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)

d. eats (Anil, Peanuts) Λ alive(Anil)

e. ∀x ¬ eats(Anil, x) V eats(Harry, x)

f. ∀x¬ [¬ killed(x)] V alive(x)

g. ∀x ¬ alive(x) V ¬ killed(x)

h. likes(John, Peanuts).

- **Rename variables or standardize variables**

a. ∀x ¬ food(x) V likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)

d. eats (Anil, Peanuts) Λ alive(Anil)

e. ∀x ¬ eats(Anil, x) V eats(Harry, x)

f. ∀x killed(x)] V alive(x)

g. ∀x ¬ alive(x) V ¬ killed(x)

h. likes(John, Peanuts).

a. ∀x ¬ food(x) V likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)

d. eats (Anil, Peanuts) Λ alive(Anil)

e. ∀w¬ eats(Anil, w) V eats(Harry, w)

f. ∀g killed(g)] V alive(g)

g. ∀k ¬ alive(k) V ¬ killed(k)

h. likes(John, Peanuts).

- **Eliminate existential instantiation quantifier by elimination.**

a. ∀x ¬ food(x) V likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)

d. eats (Anil, Peanuts) Λ alive(Anil)

e. ∀w¬ eats(Anil, w) V eats(Harry, w)

f. ∀g killed(g)] V alive(g)

g. ∀k ¬ alive(k) V ¬ killed(k)

h. likes(John, Peanuts).

There are no existential quantifiers in this question. Hence, this step is skipped.

- **Drop Universal quantifiers.**

a. ∀x ¬ food(x) V likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)

d. eats (Anil, Peanuts) Λ alive(Anil)

e. ∀w¬ eats(Anil, w) V eats(Harry, w)

f. ∀g killed(g)] V alive(g)

g. ∀k ¬ alive(k) V ¬ killed(k)

h. likes(John, Peanuts).

a. ¬ food(x) V likes(John, x)

b. food(Apple)

c. food(vegetables)

d. ¬ eats(y, z) V killed(y) V food(z)

e. eats (Anil, Peanuts)

f. alive(Anil)

g. ¬ eats(Anil, w) V eats(Harry, w)

h. killed(g) V alive(g)

i. ¬ alive(k) V ¬ killed(k)

j. likes(John, Peanuts).

a. ¬ food(x) V likes(John, x)

b. food(Apple)

c. food(vegetables)

d. ¬ eats(y, z) V killed(y) V food(z)

e. eats (Anil, Peanuts)

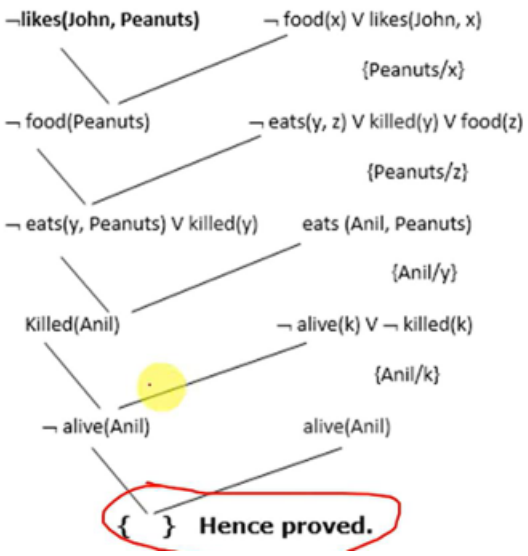
f. alive(Anil)

g. ¬ eats(Anil, w) V eats(Harry, w)

h. killed(g) V alive(g)

i. ¬ alive(k) V ¬ killed(k)

j. likes(John, Peanuts).



5.

Convert the following statement into its Equivalent Predicate Logic from

i) Marcus was a man
ii) Marcus was a Pompeian
iii) All Pompeians were Romans
iv) Caesar was a Ruler
v) All Romans were either loyal to Caesar of hated him.
vi) Everyone is loyal to someone
vii) People only try to assassinate rulers they are not loyal to.
viii) Marcus tried to assassinate Caesar.

1. Marcus was a man. — 1. man(Marcus)

2. Marcus was a Pompeian. — 2. Pompeian(Marcus)

3. All Pompeian were Romans. — 3. ∀x: Pompeian(x) → Roman(x)

4. Caesar was a ruler. — 4. ruler(Caesar)

5. All Romans were either loyal to Caesar or hated him. 5. Vx: Roman(x) → loyalto(x, Caesar) V hate(x, Caesar)

6. Everyone is loyal to someone. 6. ∀x: ∃y: loyalto(x, y)

7. People only try to assassinate rulers they are not loyal to. 7.Vx: Vy: person(x) ruler(y) tryassassinate(x, y) → -loyalto(x, y)

8. Marcus tried to assassinate Caesar. 8. tryassassinate(Marcus, Caesar)

9. All men are people 9. ∀x: man(x)→ person(x)

1. **Was Marcus Loyal to Caesar?**

Nil

↓(1)

man(Marcus)

↓(9)

person(Marcus)

↓(8)

person(Marcus)/tryassassinate(Marcus, Caesar)

↓(4)

person(Marcus)/\tryassassinate(Marcus, Caesar) /\ruler(Caesar)

↓(7, substitution)- loyalto(Marcus, Caesar)

2. Was Marcus hates Caesar?

- loyalto(Marcus, Caesar)

↓(2)

Pompeian(Marcus)

- loyalto(Marcus, Caesar)

↓(3)

Roman(Marcus)

- loyalto(Marcus, Caesar)

↓(5)

hate(Marcus, Caesar)

6. Illustrate Forward chaining & Backward Chaining in detail with example. 10 CO3

Forward chaining and backward chaining are two types of inference methods used in artificial intelligence and rule-based systems. They are techniques used to derive conclusions from a set of rules and facts. Here's a detailed explanation with examples:

Forward Chaining

Definition: Forward chaining is a data-driven approach where the inference engine starts with the available data and uses inference rules to extract more data until a goal is reached.

Process:

1. **Start with Known Facts:** Begin with the initial set of facts.
2. **Apply Rules:** Apply the inference rules to these facts.
3. **Add New Facts:** Derive new facts from the rules and add them to the set of known facts.
4. **Repeat:** Repeat the process until the goal is reached or no more rules can be applied.

Example: Consider a simple rule-based system for animal classification:

- **Rules:**
  - Rule 1: IF an animal has feathers THEN it is a bird.
  - Rule 2: IF an animal lays eggs AND has wings THEN it is a bird.
  - Rule 3: IF an animal is a bird AND cannot fly THEN it is an ostrich.
  - Rule 4: IF an animal is a bird AND can fly THEN it is a sparrow.
- **Facts:**
  - Fact 1: The animal has feathers.
  - Fact 2: The animal cannot fly.

Forward Chaining Steps:

1. **Start with Fact 1:** The animal has feathers.
2. **Apply Rule 1:** Since the animal has feathers, it is a bird.
3. **New Fact:** The animal is a bird.
4. **Combine New Fact with Fact 2:** The animal is a bird and cannot fly.
5. **Apply Rule 3:** Since the animal is a bird and cannot fly, it is an ostrich.
6. **Conclusion:** The animal is an ostrich.

Backward Chaining

Definition: Backward chaining is a goal-driven approach where the inference engine starts with a goal and works backward through inference rules to determine what facts must be true to achieve that goal.

Process:

1. **Start with Goal:** Begin with the goal or hypothesis.
2. **Determine Needed Facts:** Identify what facts or conditions must be true for the goal to be true.
3. **Find Supporting Rules:** Find rules that could produce these facts.
4. **Gather Evidence:** Work backward, gathering facts or evidence that support these rules.
5. **Repeat:** Repeat the process until the initial facts are found or the goal cannot be reached.

Example: Using the same animal classification system:

- **Goal:** Determine if the animal is an ostrich.

Backward Chaining Steps:

1. **Start with Goal:** Is the animal an ostrich?
2. **Determine Conditions:** What must be true for the animal to be an ostrich? (According to Rule 3: The animal must be a bird and cannot fly).
3. **Check Conditions:**
    - Is the animal a bird? If not, what makes it a bird? (Rule 1 or Rule 2: It must have feathers, lay eggs, and have wings).
    - Does the animal have feathers? Yes (Fact 1).
    - Can the animal not fly? Yes (Fact 2).
4. **Apply Rules:**
    - Rule 1 confirms it is a bird because it has feathers.
    - Fact 2 confirms it cannot fly.
5. **Conclusion:** Since the animal is a bird and cannot fly, it is an ostrich.

Comparison:

- **Forward Chaining:**
    - Data-driven.
    - Starts with known facts and applies rules to infer new facts.
    - Useful for situations where all data is given and we need to determine the conclusion.
    - Example applications: expert systems, production systems.
- **Backward Chaining:**
    - Goal-driven.
    - Starts with the goal and works backward to determine necessary facts.
    - Useful for diagnostic and problem-solving applications.
    - Example applications: medical diagnosis, troubleshooting systems.