UNIT–3 : Knowledge Representation (PYQ Solution)

AKTU · Artificial Intelligence (BCAI501) · Solutions by **Chirag Jain**, Ycotes

SECTION A Answers (≈70 Words Each)

Q1. Define unification in Prolog / logic programming. What is unification in AI? (Jan 2026, 2024, 2023)

Unification is the process of finding a substitution that makes two logical expressions identical. In Prolog the operator = tries to unify two terms by binding variables to values. For example, `parent(X, bob)` and `parent(tom, Y)` unify with the substitution {X/tom, Y/bob}. It is the core of pattern matching, resolution and inference in logic programming. A most general unifier (MGU) is the simplest such substitution, and unification fails if constants clash.

Q2. What is First Order Predicate Logic (FOPL)? (Jan 2025)

First Order Predicate Logic (FOPL) is a formal language that extends propositional logic by adding objects (constants and variables), functions, predicates and quantifiers. It can express properties of objects and relations among them, such as `Student(Ravi)` or `Loves(x, y)`. The quantifiers ∀ (for all) and ∃ (there exists) allow general statements, e.g. ∀x Human(x) → Mortal(x). Its syntax is precise, its semantics is clear, and it is the basis of knowledge representation and Prolog.

Q3. Write some general syntax for a Prolog program. (2023)

A Prolog program is a set of clauses ending with a full stop. Variables start with a capital letter and atoms with lowercase.

- **Fact:** `parent(tom, bob).`
- **Rule:** `grandparent(X, Z) :- parent(X, Y), parent(Y, Z).` **(`:-` means "if", the comma means "and", `;` means "or")**
- **Query:** `?- grandparent(tom, W).`

- **Comment: `% text`.**
  Facts and rules form the knowledge base, and Prolog answers queries by unification and backtracking.

Q4. Describe the process of resolution in logic programming. (2024)

Resolution is an inference rule that derives a new clause from two clauses containing complementary literals. It works by refutation: to prove a goal, add its negation to the knowledge base (in clause form) and derive the empty clause, which shows a contradiction. Steps: convert to CNF, pick clauses with a literal and its negation, unify, and form the resolvent by joining the remaining literals. Prolog uses a restricted form, SLD resolution, with backtracking.



SECTION B Answers (≈250 Words Each)

Q1. Explain the role of events, mental events, and mental objects in intelligent agents' reasoning. (Jan 2026)

To reason about a changing world and about other agents, a knowledge representation must describe **events** (things that happen), and for agents that think, **mental objects** (what is believed or known) and **mental events** (changes in those beliefs).

1. Events

- An **event** is something that happens over time and changes the state of the world, e.g. *Opening a door*, *Buying a book*.
- **Event calculus** represents time-based facts with predicates such as `Happens(e, t)`, `Initiates(e, f, t)`, `Terminates(e, f, t)` and `Holds(f, t)` (fluent f is true at time t).
- Events can be **discrete or continuous**, have **sub-events** (a trip has driving, stopping) and can be **processes** (liquid events) that have no fixed boundary.
- Role: let the agent **predict the effect of actions**, plan, and explain what has happened.

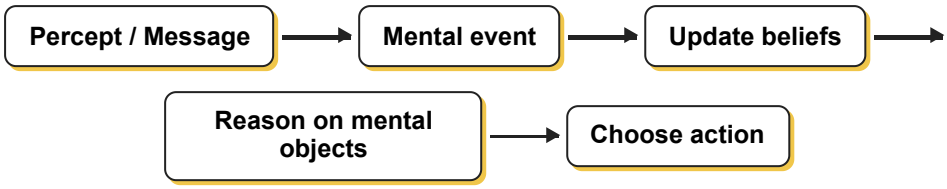
2. Mental objects

- These are the contents of an agent's mind: **beliefs, knowledge, desires, intentions**.
- Because beliefs are about **propositions**, the agent needs **reification**: a sentence is treated as a term, e.g. `Believes(Lois, Flies(Superman))`.
- **Modal logic** uses operators such as K (knows) and B (believes): K_A(P). It handles **referential opacity**: Lois believes Superman flies but does not believe Clark Kent flies, though they are the same person.
- Role: an agent can reason about **what other agents know**, which is needed in communication, negotiation and games.

3. Mental events

- A **mental event** is a change in the mental state, such as **learning a fact, forgetting, observing, deciding** or being told something.
- Example: after `Tell(Ram, "It is raining")` the event `Knows(Ram, Raining)` begins to hold.

- Role: allows the agent to **update its beliefs** (belief revision), explain behaviour and plan information-gathering actions.



Role of mental events in reasoning

Concept	Meaning	Example
Event	Occurrence in the world	Door opens
Mental object	Belief / knowledge content	Believes(Ram, Open(Door))
Mental event	Change in mental state	Ram sees the door open

Conclusion: representing events, mental objects and mental events lets an agent reason about change, about its own knowledge and about the knowledge of others.

Q2. Summarize the following sentences into symbolic (FOL) forms: (i) Everyone is loyal to someone. (ii) All Romans were either loyal to Caesar or hated him. (iii) You can fool all of the people some of the time. (iv) No purple mushroom is poisonous. (v) Everyone has a heart. (2023)

Predicates: Loyal(x, y), Roman(x), Hate(x, y), Person(x), Time(t), CanFool(x, t), Purple(x), Mushroom(x), Poisonous(x), Heart(y), Has(x, y).

(i) Everyone is loyal to someone.
$\forall x \, \exists y$ Loyal(x, y)
(For every x there is some y to whom x is loyal.)

(ii) All Romans were either loyal to Caesar or hated him.
$\forall x$ Roman(x) $\rightarrow$ Loyal(x, Caesar) $\vee$ Hate(x, Caesar)
(Use $\oplus$ for exclusive "either...or" if exactly one is meant.)

(iii) You can fool all of the people some of the time.
$\forall x \, \exists t$ Person(x) $\wedge$ Time(t) $\rightarrow$ CanFool(x, t)
(Each person can be fooled at some time.)

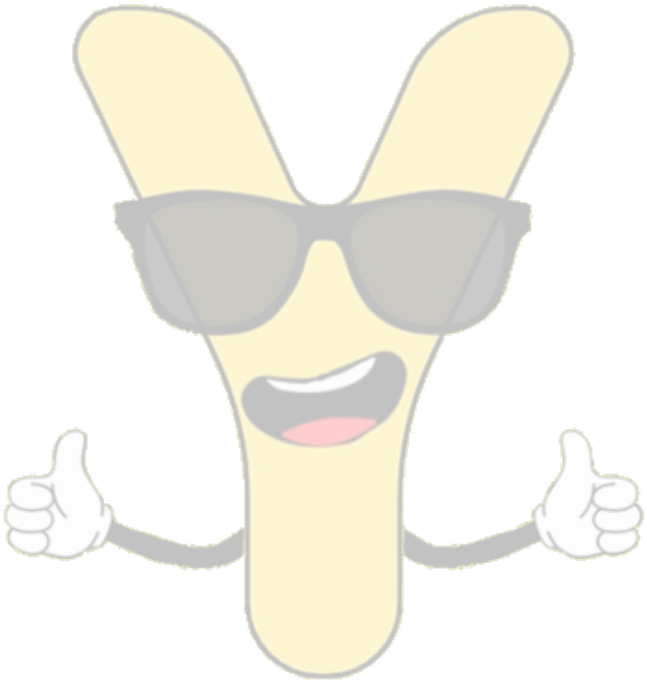
(iv) No purple mushroom is poisonous.
$\forall x$ (Purple(x) $\wedge$ Mushroom(x)) $\rightarrow \neg$Poisonous(x)
Equivalent form: $\neg \exists x$ Purple(x) $\wedge$ Mushroom(x) $\wedge$ Poisonous(x)

(v) Everyone has a heart.
$\forall x\ \text{Person}(x) \rightarrow \exists y\ \text{Heart}(y) \wedge \text{Has}(x, y)$

Points to remember

- Use $\rightarrow$ with $\forall$ and $\wedge$ with $\exists$; swapping them gives wrong meanings.
- The order of quantifiers matters: $\forall x\ \exists y\ \text{Loyal}(x, y)$ (everyone is loyal to someone, maybe different people) differs from $\exists y\ \forall x\ \text{Loyal}(x, y)$ (one person to whom everyone is loyal).
- Negation of "$\forall x\ P(x)$" is "$\exists x\ \neg P(x)$".

Conclusion: FOL lets natural-language statements be converted to precise logical sentences that an inference engine can use.



SECTION C Answers (≈500 Words Each)

Q1. Demonstrate the resolution principle by proving a simple theorem in propositional logic. (Jan 2026)

Resolution principle: from two clauses that contain a literal and its negation, we may infer a new clause (the **resolvent**) containing all the other literals.
(A ∨ B) and (¬B ∨ C) give (A ∨ C).
Resolution is **sound** and **refutation-complete**: a goal is proved by adding its negation to the knowledge base and deriving the **empty clause** (a contradiction).

Procedure

1. Write the premises in logic.
2. Convert premises and the **negated conclusion** to **CNF** (clauses).
3. Repeatedly resolve pairs of clauses having complementary literals.
4. If the empty clause □ is derived, the conclusion is proved. If no new clause can be produced, it is not provable.

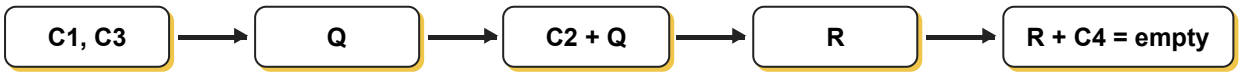
Theorem to prove (hypothetical syllogism). Premises: (1) P → Q, (2) Q → R. Prove: P → R.

Step 1: convert to CNF using A → B ≡ ¬A ∨ B.

- C1: ¬P ∨ Q
- C2: ¬Q ∨ R
- Negate the goal: ¬(P → R) ≡ ¬(¬P ∨ R) ≡ P ∧ ¬R, which gives two clauses:
- C3: P
- C4: ¬R

Step 2: resolve

Step	Clauses resolved	Resolvent
5	C1 (¬P ∨ Q) and C3 (P)	C5: Q
6	C2 (¬Q ∨ R) and C5 (Q)	C6: R
7	C6 (R) and C4 (¬R)	□ (empty clause)



Resolution refutation steps

The empty clause shows that the premises together with ¬(P → R) are contradictory, so **P → R is proved**.

Second quick example. Premises: A ∨ B, ¬A. Prove B. Negated goal: ¬B. Clauses: A ∨ B, ¬A, ¬B. Resolve (A ∨ B) with ¬A to get B; resolve B with ¬B to get □. Hence B is true.

Advantages and limits

- Needs only one inference rule, so it is easy to automate; it is the basis of Prolog and theorem provers.
- In first-order logic it needs **unification** and Skolemisation, and the search can grow large; strategies like **unit preference** and **set of support** reduce it.

Conclusion: by negating the goal and deriving the empty clause, resolution gives a mechanical proof of the theorem.

Q2. Create a forward chaining example in a rule-based system to determine eligibility for a scholarship. (Jan 2026)

Forward chaining is **data-driven** reasoning: starting from known facts, the inference engine repeatedly fires rules whose conditions are satisfied and adds their conclusions to the facts until no new fact can be derived or the goal is reached.

Knowledge base (rules)

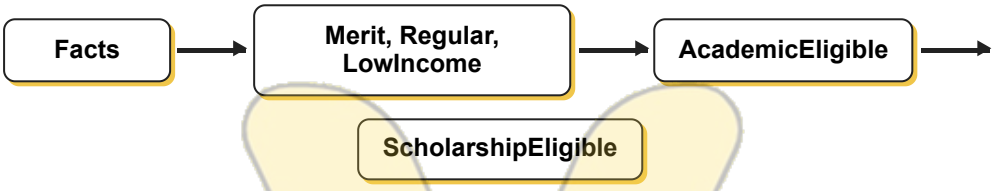
- R1: IF marks ≥ 75 THEN Merit
- R2: IF attendance ≥ 75 THEN RegularStudent
- R3: IF family income ≤ 3,00,000 THEN LowIncome
- R4: IF Merit AND RegularStudent THEN AcademicEligible
- R5: IF AcademicEligible AND LowIncome THEN ScholarshipEligible
- R6: IF Disciplinary action THEN NOT ScholarshipEligible

Initial facts (applicant Ravi): marks = 82, attendance = 88, family income = 2,40,000, no disciplinary action.

Execution trace

Cycle	Rule fired	Condition check	New fact added
1	R1	82 ≥ 75 true	Merit
1	R2	88 ≥ 75 true	RegularStudent
1	R3	2,40,000 ≤ 3,00,000 true	LowIncome
2	R4	Merit and RegularStudent true	AcademicEligible
3	R5	AcademicEligible and LowIncome true	ScholarshipEligible
4	R6	no disciplinary action, not applicable	none

No more rules fire, so the engine stops with **Ravi is eligible for the scholarship**.



Forward chaining chain of inference

Another applicant (Meena): marks = 70, attendance = 90, income = 2,00,000. R1 does not fire (70 < 75), R2 and R3 fire, but R4 needs Merit, so R4 and R5 do not fire. Result: **not eligible**.

Architecture used: working memory (facts), rule base, inference engine with conflict resolution (choose among several applicable rules, e.g. by priority or order).

Conclusion: forward chaining naturally suits eligibility checks, where the data of the applicant is known and the system finds what can be concluded.

Q3. Compare semantic networks and frames as knowledge representation methods. (Jan 2025)

Both are **structured (non-logical)** knowledge representation schemes that organise knowledge about objects and relations.

Semantic network

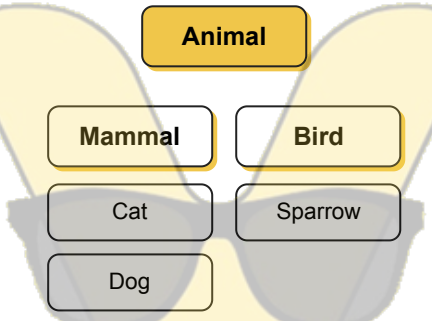
- A **graph** whose **nodes** are objects or concepts and **labelled arcs** are relations (is-a, has-part, instance-of).
- Supports **inheritance**: a node inherits the properties of its parents. Example: Tom –instance-of→ Cat –is-a→ Mammal –is-a→ Animal; "Animals breathe" applies to Tom.

- Inference is by **traversing links**.

Frame

- A **record-like data structure** for a stereotyped situation or object, with **slots** (attributes), **facets** (type, default, range) and **values**.
- Features: **default values**, **procedural attachments** (if-needed, if-added demons) and inheritance through **is-a** hierarchies.
- Example frame:

```
Frame: Car
  is-a: Vehicle
  wheels: 4 (default)
  engine: [petrol / diesel]
  if-needed: compute-mileage()
```



Is-a hierarchy used in both methods

Comparison

Basis	Semantic network	Frame
Structure	Graph of nodes and links	Collection of slots and fillers
Representation	Binary relations	Objects with attributes
Defaults	Weak	Strong (default values)
Procedures	Not included	If-needed / if-added attachments
Inheritance	Yes via is-a links	Yes via is-a and slots
Readability	Visual and intuitive	Compact and organised
Handling complex objects	Becomes cluttered	Handles well
Semantics	Informal, ambiguous links	Better defined
Use	Concept relations, WordNet	Expert systems, object-oriented design

Similarities: both support inheritance and a hierarchy; frames can be seen as a more developed semantic network.

Limitations: both lack clear formal semantics, cannot easily express negation, quantifiers or disjunction, and exceptions to inheritance cause problems.

Conclusion: semantic networks are best for showing relations among concepts, frames for describing detailed structured objects with defaults and procedures.

Q4. Explain how FOPL is used for representing knowledge in AI systems and how it is utilised in Prolog. (Jan 2025, 2024)

First-order predicate logic (FOPL) is a formal language to represent facts about objects, their properties and relations, so that a machine can reason on them.

Syntax elements

- **Constants:** Ram, 5, Delhi. **Variables:** x, y. **Functions:** FatherOf(x).
- **Predicates:** Student(x), Loves(x, y). **Connectives:** ¬, ∧, ∨, →, ↔. **Quantifiers:** ∀ (universal), ∃ (existential). **Equality:** =.
- **Terms** denote objects; **atomic sentences** are predicates applied to terms; **complex sentences** use connectives and quantifiers.

Representing knowledge: examples

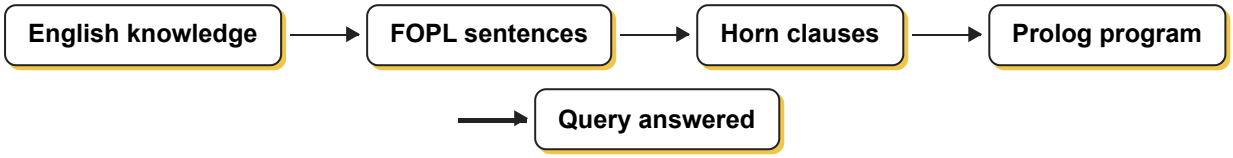
English	FOPL
Ram is a student	Student(Ram)
All students are intelligent	∀x Student(x) → Intelligent(x)
Some student likes AI	∃x Student(x) ∧ Likes(x, AI)
No student is perfect	¬∃x Student(x) ∧ Perfect(x)
Ram is Sita's father	Father(Ram, Sita)

Inference: from ∀x Man(x) → Mortal(x) and Man(Socrates), universal instantiation and Modus Ponens give Mortal(Socrates). Proof uses **unification, forward/backward chaining and resolution**.

Use in Prolog

- Prolog is based on a subset of FOPL called **Horn clauses** (at most one positive literal).

- A fact `student(ram).` is an atomic sentence; a rule `mortal(X) :- man(X).` represents ∀x Man(x) → Mortal(x); variables are implicitly universally quantified.
- A **query** `?- mortal(socrates).` is proved by **SLD resolution** with unification and **backtracking**.
- Example: `parent(tom, bob). parent(bob, ann). grandparent(X, Z) :- parent(X, Y), parent(Y, Z).` and the query `?- grandparent(tom, Who).` gives `Who = ann`.



From knowledge to Prolog

Advantages: expressive, precise, sound and complete inference, declarative.
Limitations: cannot represent uncertainty; inference is semi-decidable and can be slow; Prolog cannot express disjunctive or negative facts directly (it uses negation as failure).

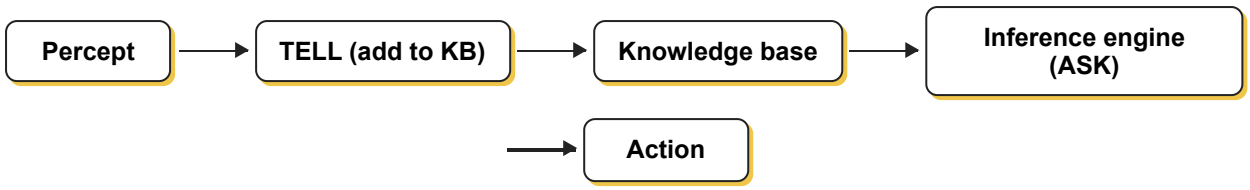
Conclusion: FOPL gives a clear way to encode knowledge, and Prolog turns its Horn-clause subset into an executable programming language.

Q5. Explore the knowledge-based agent with a diagram. How does the inference engine contribute to learning? (2023)

A **knowledge-based agent (KBA)** maintains an explicit **knowledge base (KB)** of sentences in a formal language and decides what to do by reasoning on it. It is **declarative**: what the agent knows can be changed by telling it new facts, without reprogramming.

Components

- **Knowledge base:** the set of known sentences (background knowledge plus percepts).
- **Inference engine:** domain-independent algorithms that derive new sentences from the KB (forward chaining, backward chaining, resolution).
- **TELL / ASK interface:** TELL adds a sentence to the KB; ASK queries it.



Knowledge-based agent

Agent program (each time step)

1. TELL the KB what the agent perceived.
2. ASK the KB what action it should do.
3. TELL the KB that the action was executed; return the action.

Levels: at the **knowledge level** we specify what the agent knows and its goals; at the **implementation level** we choose data structures and algorithms.

Example: wumpus world. The agent's KB holds rules like "a pit causes breezes in adjacent squares". From percepts (breeze at [1,2]) the inference engine deduces which squares are safe or dangerous and selects a safe move.

How the inference engine contributes to learning

- It **derives new knowledge** (implicit consequences) from existing facts, and these conclusions are added to the KB, so the agent's knowledge grows.
- It performs **deduction, abduction and induction**-style reasoning to **update beliefs** when new percepts arrive.
- It **detects contradictions**, leading to **belief revision**.
- In **explanation-based learning**, it builds a proof for an example and generalises it into a new rule.
- It supports **planning by prediction**: simulated outcomes of actions improve future decisions.
- In **inductive logic programming**, hypotheses are tested by inference against examples.

Part	Role
KB	Stores knowledge
Inference engine	Reasons and derives new facts
Learning element	Adds or revises rules using derived results

Conclusion: the KB stores what the agent knows, and the inference engine turns it into new facts and decisions, which is how the agent keeps improving.

Q6. "As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen." Justify "Robert is criminal" using forward chaining or backward chaining. (2023)

Step 1: convert the statements into FOL (Horn clauses)

1. American(p) ∧ Weapon(q) ∧ Sells(p, q, r) ∧ Hostile(r) → Criminal(p)
2. Owns(A, T1) (country A has some missiles; T1 is a missile)
3. Missile(T1)
4. Missile(p) ∧ Owns(A, p) → Sells(Robert, p, A) (all missiles were sold by Robert)
5. Missile(p) → Weapon(p)
6. Enemy(p, America) → Hostile(p)
7. American(Robert)
8. Enemy(A, America)

Step 2: forward chaining (start from facts)

Known facts: Owns(A,T1), Missile(T1), American(Robert), Enemy(A,America).

Iteration	Rule used	New fact derived
1	Rule 5 with p = T1	Weapon(T1)
1	Rule 4 with p = T1	Sells(Robert, T1, A)
1	Rule 6 with p = A	Hostile(A)
2	Rule 1 with p = Robert, q = T1, r = A	Criminal(Robert)

All of American(Robert), Weapon(T1), Sells(Robert,T1,A) and Hostile(A) are now true, so Rule 1 fires and **Criminal(Robert)** is added. The goal is proved.



Forward chaining proof

Step 3: backward chaining (start from the goal), as a check

- Goal: Criminal(Robert). By rule 1 it needs sub-goals: American(Robert), Weapon(q), Sells(Robert,q,r), Hostile(r).
- American(Robert): fact 7. ✓

- Weapon(q): via rule 5 needs Missile(q); fact 3 gives q = T1. ✓
- Sells(Robert, T1, r): via rule 4 needs Missile(T1) ✓ and Owns(A, T1) ✓, so r = A. ✓
- Hostile(A): via rule 6 needs Enemy(A, America), fact 8. ✓
  All sub-goals succeed, so Criminal(Robert) is true.

Conclusion: either method proves that Robert has broken the law and is a **criminal**; forward chaining proceeds from the facts, backward chaining from the goal.

Q7. Compare and contrast forward chaining and backward chaining in rule-based reasoning systems, with examples. (2024)

A rule-based system has a **knowledge base of IF-THEN rules**, a **working memory of facts** and an **inference engine**. The engine can reason in two directions.

Forward chaining (data-driven)

- Starts with known facts, finds rules whose IF part is satisfied, fires them and adds the THEN part to the facts. Continues until the goal is derived or nothing more is possible.
- Example: Rules: R1 $A \wedge B \rightarrow C$, R2 $C \rightarrow D$. Facts A, B. Fire R1 to get C, fire R2 to get D.
- Suits monitoring, planning, configuration (e.g. XCON) and cases where many conclusions are wanted from given data.

Backward chaining (goal-driven)

- Starts with a **goal (hypothesis)**, finds rules that conclude it, and treats their IF parts as new sub-goals; stops when sub-goals match known facts.
- Example: Goal D. R2 needs C. R1 needs A and B, and both are facts, so D is proved.
- Used in Prolog, diagnosis (MYCIN) and question answering.



Forward chaining direction



Backward chaining direction

Comparison

Basis	Forward chaining	Backward chaining
Direction	Facts → conclusion	Goal → facts
Driven by	Data	Goal
Strategy	Bottom-up	Top-down
Search	Breadth-first typical	Depth-first typical
Efficiency	May derive many irrelevant facts	Examines only relevant rules
Best when	Many goals, data arrive continuously	One specific goal or query
Example systems	OPS5, CLIPS, monitoring	Prolog, MYCIN
Weakness	Wasted work	May loop or repeat sub-goals

Medical example. Facts: fever, rash. Forward: rules lead to "measles suspected" then "recommend isolation". Backward: goal "Is it measles?" asks for fever and rash and checks them.

Common points: both use Modus Ponens, are sound, and are complete for Horn clauses. Some systems combine both (bidirectional).

Conclusion: choose forward chaining when facts are known and conclusions are open, and backward chaining when a specific goal is to be verified.

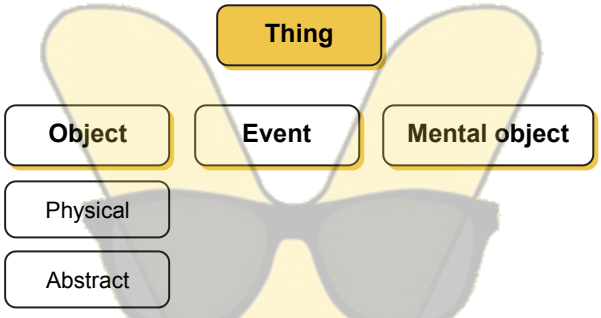
Q8. Explain the categories and objects in ontological engineering with examples. How is knowledge represented in ontological engineering, and what role does it play in building intelligent systems? (Jan 2025, 2024)

Ontological engineering is the process of creating **ontologies**: formal, shared descriptions of the concepts in a domain and their relationships, used to represent general knowledge such as **events, time, physical objects and beliefs**. An **upper ontology** is a general framework (objects, categories, events, time, mental objects) on which domain-specific ontologies are built.

Categories and objects

- **Objects** are individual things, e.g. Tom, Basketball1.
- **Categories** group objects with common properties, e.g. Cat, Basketballs. Categories serve as **predicates** (Cat(Tom)) or can be **reified as objects** (Member(Tom, Cats)).

- **Relations among categories**
- **Subcategory:** Subset(Cats, Mammals).
- **Inheritance:** all members of Mammals inherit the properties of the category.
- **Disjoint:** Disjoint({Animals, Vegetables}).
- **Exhaustive decomposition / partition:** Americans, Canadians, Mexicans partition North Americans.
- **Physical composition:** PartOf(Bucharest, Romania).
- **Measurements:** Length(L1) = Inches(1.5).
- **Substances and objects:** "stuff" like water (mass nouns) differs from countable "things".
- **Events and time:** event calculus, intervals.
- **Mental objects:** beliefs and knowledge of agents.



Part of an upper ontology

Knowledge representation methods in ontologies

- **First-order logic** axioms for categories and relations.
- **Semantic networks / graphs** and **frames**.
- **Description logics** and web standards **RDF, RDFS and OWL**, with classes, properties and instances.
- **Taxonomic hierarchies** (is-a), as in WordNet and Cyc.

Example (OWL-style): Class Student ⊆ Person; property enrolledIn(Student, Course).

Role in building intelligent systems

- Provides a **shared vocabulary** so agents and systems can understand each other (interoperability).
- Enables **reasoning**, such as classification and consistency checking.
- Basis of the **Semantic Web, knowledge graphs, expert systems, chatbots** and medical coding systems.
- Supports **knowledge reuse** and easier maintenance.

- Helps in **natural language understanding** and **data integration**.

Item	Example
Object	Tom
Category	Cat
Subcategory	Cat ⊂ Mammal
Relation	PartOf(Wheel, Car)

Limits: building ontologies is costly, categories have fuzzy boundaries, and different groups may disagree on concepts.

Conclusion: ontological engineering organises knowledge into categories, objects and relations, giving intelligent systems a structured base to reason from.

