UNIT 3

INT426:GENERATIVE ARTIFICIAL INTELLIGENCE

Big Ideas

- Logic is a great knowledge representation language for many AI problems
- **Propositional logic** is the simple foundation and fine for some AI problems
- **First order logic** (FOL) is much more expressive as a KR language and more commonly used in AI
- There are many variations: horn logic, higher order logic, three-valued logic, probabilistic logics, etc.

Propositional logic in Artificial intelligence

- Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions/ is a statement of a fact.

- Propositional logic is also called Boolean logic as it works on 0 and 1.
- In propositional logic, we use symbolic variables to represent the logic, and we can use any symbol for a representing a proposition, such A, B, C, P, Q, R, etc.
- Propositions can be either true or false, but it cannot be both.
- Statements such as "**Where is Ajay**", "**How are you**", "**What is your name**", are not propositions.

1.Atomic Propositions: are the simple propositions. It consists of a single proposition symbol.

2+1 is 3, it is an atomic proposition as it is a **true** fact.

"The Sun is cold" is also a proposition as it is a **false** fact.

2. Compound propositions: Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.

a) "It is raining today, and street is wet."

b) "Ankit is a doctor, and he play circket."

Sentences are combined by **connectives**:

$\wedge$	and	[conjunction/ Logical AND]
$\vee$	or	[disjunction/ Logical OR]
$\Rightarrow$	implies	[implication / conditional/ Material conditional operator]
$\Leftrightarrow$	is equivalent	[biconditional/ if and only if(iff)/ material biconditional operator]
$\neg$	not	[negation/ logical negation]

Literal: atomic sentence or negated atomic sentence

$P, \neg P$

Examples of PL sentences

- $(P \wedge Q) \rightarrow R$

  "If it is hot and humid, then it is raining"

- $Q \rightarrow P$

  "If it is humid, then it is hot"

- Q

  "It is humid."

- We're free to choose better symbols, btw:

  Ho = "It is hot"

  Hu = "It is humid"

  R = "It is raining"

Propositional logic (PL)

- Simple language for showing key ideas and definitions
- User defines set of propositional symbols, like P and Q
- User defines **semantics** of each propositional symbol:
  - P means “It is hot”, Q means “It is humid”, etc.
- A sentence (well formed formula) is defined as follows:
  - A symbol is a sentence
  - If S is a sentence, then $\neg$S is a sentence
  - If S is a sentence, then (S) is a sentence
  - If S and T are sentences, then (S $\vee$ T), (S $\wedge$ T), (S $\rightarrow$ T), and (S $\leftrightarrow$ T) are sentences
  - A sentence results from a finite number of applications of the rules

Predicate Logic

Quantifiers

1) Universal quantification

(∀x)P(x) means that P holds for all values of x in the domain associated with that variable

E.g., (∀x) dolphin(x) → mammal(x)

2) Existential quantification

(∃ x)P(x) means that P holds for some value of x in the domain associated with that variable

E.g., (∃ x) mammal(x) ∧ lays-eggs(x)

The predicates are chosen in such a way that they represent the main theme of statement.

Predicate Calculus Sentences

An atomic sentence is a predicate constant of arity n followed by n terms t_1, t_2, t_3,..... t_n, enclosed in a parenthesis and separated by commas. Every atomic sentence is a sentence. Predicate calculus sentences are defined as:

1. The truth values 'true' and 'false' are atomic sentences.
2. If **s** is a sentence then its negation $\neg$**s** is also a sentence.
3. If $\mathbf{s}_1$ and $\mathbf{s}_2$ are sentences then their conjunction $\mathbf{s}_1 \wedge \mathbf{s}_2$ is also a sentence.
4. If $\mathbf{s}_1$ and $\mathbf{s}_2$ are sentences then their disjunction $\mathbf{s}_1 \vee \mathbf{s}_2$ is also a sentence.
5. If $\mathbf{s}_1$ and $\mathbf{s}_2$ are sentences then their implication $\mathbf{s}_1 \rightarrow \mathbf{s}_2$ is also a sentence.
6. If $\mathbf{s}_1$ and $\mathbf{s}_2$ are sentences then their equivalence $\mathbf{s}_1 \equiv \mathbf{s}_2$ is also a sentence.
7. If **X** is a variable and **s** is a sentence then $\exists$**Xs** is also a sentence.
8. If **X** is a variable and **s** is a sentence then $\forall$**Xs** is also a sentence.

A Predicate Logic Example

1. Marcus was a man.
2. Marcus was a Pompeian.
3. All Pompeian were Romans.
4. Caesar was a ruler.
5. All Romans were either loyal to Caesar or hated him.
6. Everyone is loyal to someone.
7. People only try to assassinate rulers they are not loyal to.
8. Marcus tried to assassinate Caesar.

Marcus was a man.

man(Marcus)

2. Marcus was a Pompeian.

Pompeian(Marcus)

3. All Pompeians were Romans.

∀x: Pompeian(x) → Roman(x)

4. Caesar was a ruler.

ruler(Caesar)

5. All Pompeians were either loyal to Caesar or hated him.

- inclusive-or

∀x: Roman(x) → loyalto(x, Caesar) ∨ hate(x, Caesar)

- exclusive-or

∀x: Roman(x) → (loyalto(x, Caesar) ∧¬ hate(x, Caesar)) ∨(¬loyalto(x, Caesar) ∧ hate(x, Caesar))

6. Everyone is loyal to someone.

∀x: ∃y: loyalto(x, y)

7. People only try to assassinate rulers they are not loyal to.

∀x: ∀y: person(x) ∧ ruler(y) ∧ tryassassinate(x, y) → ¬loyalto(x, y)

8. Marcus tried to assassinate Caesar.

tryassassinate(Marcus, Caesar)

Resolution in FOL

- Resolution is a theorem proving procedure that proceeds by building refutation proofs, i.e., proofs by contradictions.
- In refutation, initially it presumes that statement to be proved is not true, and then it proves that whatever we have assumed is not true, and hence the original statement is true.
- Resolution is applicable only on the facts represented in clausal form.
- Clause form: it is disjunction of literals.

Steps for Resolution:

1. Conversion of facts into first-order logic.
2. Convert FOL(Predicate Logic) statements into CNF(Conjunctive Normal Form)
3. Negate the statement which needs to prove (proof by contradiction)
4. Draw resolution graph (unification).

Conversion of FOL into CNF

Step 1: Eliminate implication and equivalence using replacement. $P \rightarrow Q$ is replaced by $\neg P \vee Q$ and $P \leftrightarrow Q$ is replaced by $(\neg P \vee Q) \wedge (\neg Q \vee P)$.

Step 2: Move all negations to immediately precede an atom i.e., replace $\neg(\neg P)$ by P and use De Morgan's laws for other replacements i.e., $\neg(a \wedge b) = \neg a \vee \neg b$ and $\neg(a \vee b) = \neg a \wedge \neg b$. Replace $\exists x: P(x)$ by $\forall x: \neg P(x)$ and $\neg(\forall x: P(x)) = \exists x: \neg P(x)$ i.e., use existential quantifier for removing negative literal and universal quantifier.

Step 3: Introduce dummy variable to segregate binding of variables, e.g., the expression $\forall x: P(x) \rightarrow (\exists x (Q(x))$ is rewritten using a new variable as $\forall x: P(x) \rightarrow (\exists y(Q(y))$. Perform the introduction of new variable throughout in the FOPL statement.

Step 4: Eliminate existential quantifiers by introducing new constants. Conceptually existential quantifier indicates the presence of one specific value for some FOPL. This is replaced by a constant, e.g., statement $(\exists y(\text{principal}(y))$ is transformed to statement principal (name), where variable holds the value of the real name of the principal. This process of removing existential quantifier is called as skolemisation. If the existential quantifier falls in the scope of universal quantifier then value that satisfies predicate depends upon the value of universal variable, e.g.,

$$\forall x: \exists y : \text{student}(y, x)$$

That means for all student x there will be one teacher y. The same concept can also be written as $\forall x$: student (teacher(x), x).

Teacher (x) is newly introduced predicate, which means teacher of x. as for every student a teacher is must, and for every student there will

be a unique teacher, so instead of calling that variable 'x' directly, call it teacher(x). The newly introduced predicate is called *skolem function*. Variables are removed using skolem function.

Step 5: Remove all universal quantifiers and bring the expression in conjunctive normal form. That is remove 'OR' by using theorem $a \vee (b \vee c) = (a \vee b) \vee c$ and remove 'AND' by using distributive property i.e. $(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)$

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.
   Prove by resolution that:
f. John likes peanuts.

Step-1:

a. $\forall x$: food(x) $\rightarrow$ likes(John, x)

b. food(Apple) $\wedge$ food(vegetables)

c. $\forall x \forall y$: eats(x, y) $\wedge \neg$ killed(x) $\rightarrow$ food(y)

d. eats (Anil, Peanuts) $\wedge$ alive(Anil).

e. $\forall x$: eats(Anil, x) $\rightarrow$ eats(Harry, x)

f. $\forall x$: $\neg$ killed(x) $\rightarrow$ alive(x) } **added predicates.**

g. $\forall x$: alive(x) $\rightarrow \neg$ killed(x) }

h. likes(John, Peanuts)

Step-2: Conversion of FOL into CNF

Eliminate all implication (→)

1.∀x ¬ food(x) V likes(John, x)
2.food(Apple) Λ food(vegetables)
3.∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)
4.eats (Anil, Peanuts) Λ alive(Anil)
5.∀x ¬ eats(Anil, x) V eats(Harry, x)
6.∀x¬ [¬ killed(x)] V alive(x)
7.∀x ¬ alive(x) V ¬ killed(x)
8.likes(John, Peanuts).

Move negation (¬)inwards

1. ∀x ¬ food(x) V likes(John, x)
2. food(Apple) Λ food(vegetables)
3. ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)
4. eats (Anil, Peanuts) Λ alive(Anil)
5. ∀x ¬ eats(Anil, x) V eats(Harry, x)
6. ∀x killed(x)] V alive(x)
7. ∀x ¬ alive(x) V ¬ killed(x)
8. likes(John, Peanuts).

Rename variables or standardize variables

1. ∀x ¬ food(x) V likes(John, x)
2. food(Apple) Λ food(vegetables)
3. ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)
4. eats (Anil, Peanuts) Λ alive(Anil)
5. ∀w¬ eats(Anil, w) V eats(Harry, w)
6. ∀g killed(g)] V alive(g)
7. ∀k ¬ alive(k) V ¬ killed(k)
8. likes(John, Peanuts).

Eliminate existential instantiation quantifier by elimination.

Drop Universal quantifiers.

Distribute conjunction ∧ over disjunction ¬

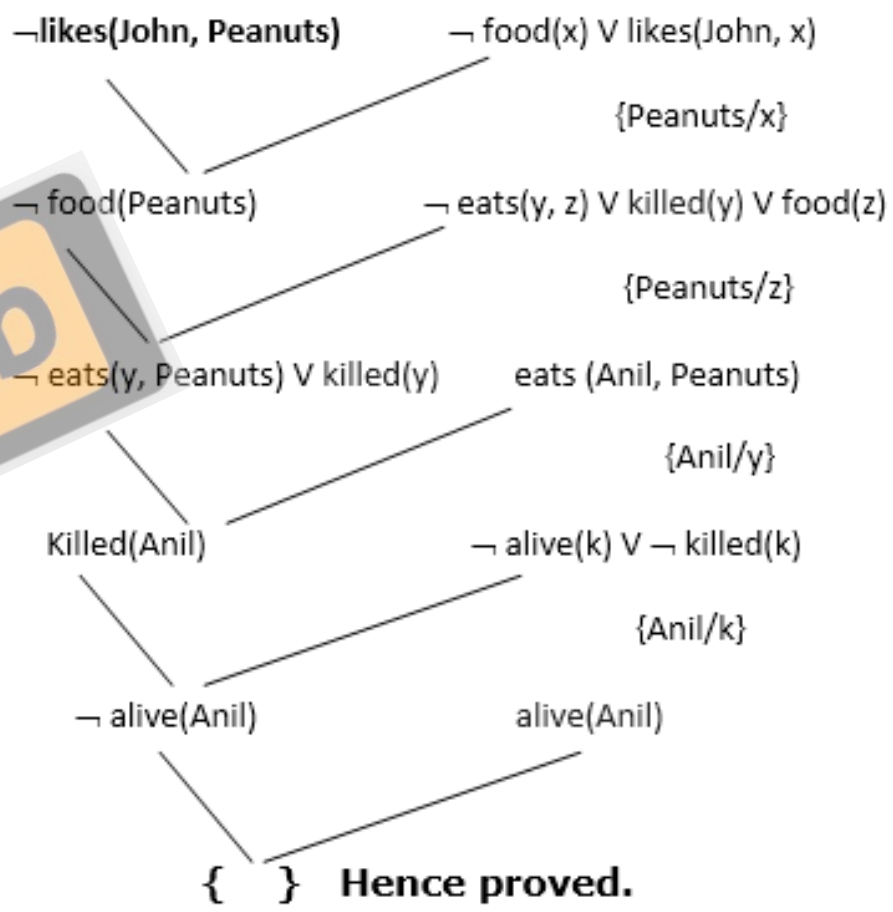
Step-3: Negate the statement to be proved

In this statement, we will apply negation to the conclusion statements, which will be written as **¬likes(John, Peanuts)**

Drop Universal quantifiers.

1. ¬ food(x) V likes(John, x)
2. food(Apple)
3. food(vegetables)
4. ¬ eats(y, z) V killed(y) V food(z)
5. eats (Anil, Peanuts)
6. alive(Anil)
7. ¬ eats(Anil, w) V eats(Harry, w)
8. killed(g) V alive(g)
9. ¬ alive(k) V ¬ killed(k)
10. likes(John, Peanuts).

Step-4: Draw Resolution graph:



What is logic programming

- Logic programming is a programming paradigm, which basically means it is a particular way to approach programming.
- The concept of programming paradigms arises owing to the need to classify programming languages. It refers to the way computer programs solve problems through code.
- Some programming paradigms are primarily concerned with implications or the sequence of operations used to achieve the result.
- Other programming paradigms are concerned about how we organize the code.

The most popular logic programming system is PROLOG. A PROLOG program is described as a series of logical assertions , each which is a Horn Clause. A 'horn clause' is a clause having at most one positive literal.

Some of the more popular programming paradigms:

Imperative: This uses statements to change a program's state, thus allowing for side effects.
Functional: This treats computation as an evaluation of mathematical functions and does not allow changing states or mutable data.
Declarative: This is a way of programming where you write your programs by describing what you want to do and not how you want to do it. You express the logic of the underlying computation without explicitly describing the control flow.

Object Oriented: This groups the code within the program in such a way that each object is responsible for itself. The objects contain data and methods that specify how the changes happen.
Procedural: This groups the code into functions and each function is responsible for a particular series of steps.
Symbolic: This uses a particular style of syntax and grammar through which the program can modify its own components by treating them as plain data.
Logic: This views computation as automatic reasoning over a database of knowledge consisting of facts and rules.

- In order to understand logic programming, we need to understand the concepts of computation and deduction.
- To compute something, we start with an expression and a set of rules. This set of rules is basically the program. We use these expressions and rules to generate the output.

- On the other hand, if we want to deduce something, we need to start from a conjecture. We then need to construct a proof according to a set of rules.

The process computation is mechanical, whereas the process of deduction is more creative.

When we write a program in the logic programming paradigm, we specify a set of statements based on facts and rules about the problem domain and the solver solves it using this information.

Understanding the building blocks of logic programming

- In programming object-oriented or imperative paradigms, we always have to specify how a variable is defined.
- In logic programming, things work a bit differently. We can pass an uninstantiated argument to a function and the interpreter will instantiate these variables for us by looking at the facts defined by the user.

✓ We need to specify something called relations in logic programming. These relations are defined by means of clauses called facts and rules.
✓ Every logic program needs facts to work with, so that it can achieve the given goal based on them.

✓ Rules are the things we have learned about how to express various facts and how to query them. They are the constraints that we have to work with and they allow us to make conclusions about the problem domain.
✓ For example, let's say you are working on building a chess engine. You need to specify all the rules about how each piece can move on the chessboard.

Forward and Backward Chaining in Artificial Intelligence

- Backward and forward chaining are methods of reasoning that exist in the Expert System Domain of artificial intelligence.
- These techniques are used in expert systems such as MYCIN and DENDRAL to generate solutions to real life problems.

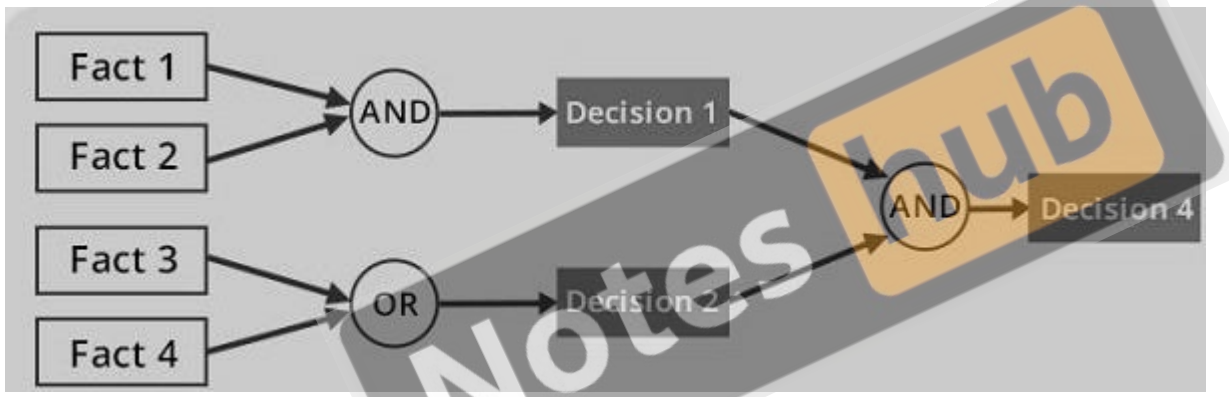
- An expert system is a computer application that uses rules, approaches, and facts to provide solutions to complex problems.
- Examples of expert systems include MYCIN and DENDRAL.
- MYCIN uses the backward chaining technique to diagnose bacterial infections.
- DENDRAL employs forward chaining to establish the structure of chemicals.

There are three components in an expert system: **user interface, inference engine, and knowledge base**. The user interface enables users of the system to interact with the expert system. High-quality and domain-specific knowledge is stored in the knowledge base.

Backward and forward chaining stem from the inference engine component. This is a component in which logical rules are applied to the knowledge base to get new information or make a decision. The backward and forward chaining techniques are used by the inference engine as strategies for proposing solutions or deducing information in the expert system.

Forward chaining

Forward chaining is a method of reasoning in artificial intelligence in which inference rules are applied to existing data to extract additional data until an endpoint (goal) is achieved.



In this type of chaining, the inference engine starts by evaluating existing facts, derivations, and conditions before deducing new information. An endpoint (goal) is achieved through the manipulation of knowledge that exists in the knowledge base.

Forward chaining can be used in planning, monitoring, controlling, and interpreting applications.

Properties of forward chaining

- The process uses a down-up approach (bottom to top).
- It starts from an initial state and uses facts to make a conclusion.
- This approach is data-driven.
- It's employed in expert systems and production rule system.

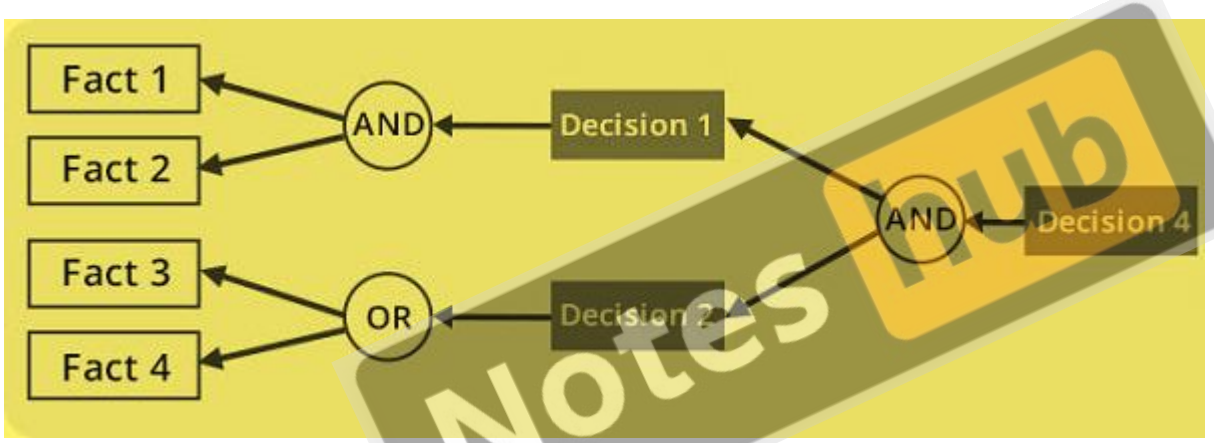
Examples

A
A->B
B
A is the starting point. A->B represents a fact. This fact is used to achieve a decision B.

Tom is running (A)
If a person is running, he will sweat (A->B)
Therefore, Tom is sweating. (B)

Backward chaining

Backward chaining is a concept in artificial intelligence that involves backtracking from the endpoint or goal to steps that led 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 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

•It’s a goal-driven method of reasoning.
•The endpoint (goal) is subdivided into sub-goals to prove the truth of facts.
•A backward chaining algorithm is employed in inference engines, game theories, and complex database systems.

Example

B
A->B
A
B is the goal or endpoint, that is used as the starting point for backward tracking. A is the initial state. A->B is a fact that must be asserted to arrive at the endpoint B.

Tom is sweating (B).
If a person is running, he will sweat (A->B).
Tom is running (A).


Notes
hub

Knowledge Representation

✓Earlier it was believed that the best approach to solutions was through the development of General Purpose Problem Solver, that is, systems powerful to prove a theorem in geometry, perform a complex robotic task, or to develop a plan to complete a sequence of operations.

✓But it was deduced that the systems became effective only when the solution methods incorporated domain specific rules and facts, i.e. after gaining specific knowledge.

✓It eventually led to knowledge based systems.

Knowledge Representation

✓**Importance of Knowledge**

✓Knowledge can be defined as the body of facts and principles accumulated by human-kind or the act, fact or state of knowing.

✓In actuality it is more than this, it also includes having a familiarity with language, concepts, procedures, rules, ideas, abstractions , places, customs, facts associations along with ability to use these notions effectively in modeling different aspects of the world.

✓**How is knowledge stored in biological organisms and computers?**

✓ Human brain weighs 3.3 pounds - estimated number of neurons 10^{12} - potential storage – 10^{14}

✓In computers, knowledge is also stored as symbolic structures, in the form of collections of magnetic spots and voltage states.

Knowledge Representation

- ✓Let’s take the following examples
  - ✓A is tall.
  - ✓A Loves B. ☺
  - ✓C has learned to use recursion to manipulate linked lists in several programming languages.

  - ✓1st one represents a fact, an attribute possessed by a person.
  - ✓2nd expresses a complex binary relation between two persons.
  - ✓3rd is most complex, expressing relations between a person and more abstract programming concepts.

Knowledge Representation

The basic components of knowledge are:

- A set of data
- A form of belief or hypothesis
- A kind of information

✓We should not confuse Knowledge with data. Knowledge is different from data. Data is raw form of observations. *Knowledge is organized form of data and procedures which can be used for some useful purposes.*

– Physician example

✓Belief v/s Hypothesis

✓Belief is any meaningful and coherent expression that can be represented. Thus belief may be true or false.

✓Hypothesis is a justified belief that is not known to be true.

✓Epistemology: study of nature of knowledge

✓Meta knowledge: is knowledge about knowledge(knowledge about what we know)

Knowledge Representation

✓Knowledge can be of following types

✓**Declarative (statements):** it is the knowledge which gives the simple facts about any organization. Declarative knowledge means representation of facts. This tells 'what' about a situation.

- ✓ Static facts e.g location of college building
- ✓ Dynamic facts: e.g new courses may be added

Declarative knowledge does not tell anything regarding functioning of the concerned objects.

✓**Procedural (facts):** the procedural knowledge represents the functioning of the organization. It describes dynamic attributes using production rules.

If :

student has deposited fees

student has opted a courses and student has attended 90% classes and

student has passed examination

Then: declare the student pass

The procedural knowledge in AI program is represented as production rules.

✓**Heuristics (rule of thumb / experience):** this type knowledge can be defined as experimental , rarely discussed and individualistic knowledge.

✓**Structural Knowledge** – It is a basic problem-solving knowledge that describes the relationship between concepts and objects.

✓**Meta Knowledge –** Meta Knowledge defines knowledge about other types of Knowledge.

Knowledge Characteristics

A good system for the representation of knowledge in a particular domain should posses the following four properties:

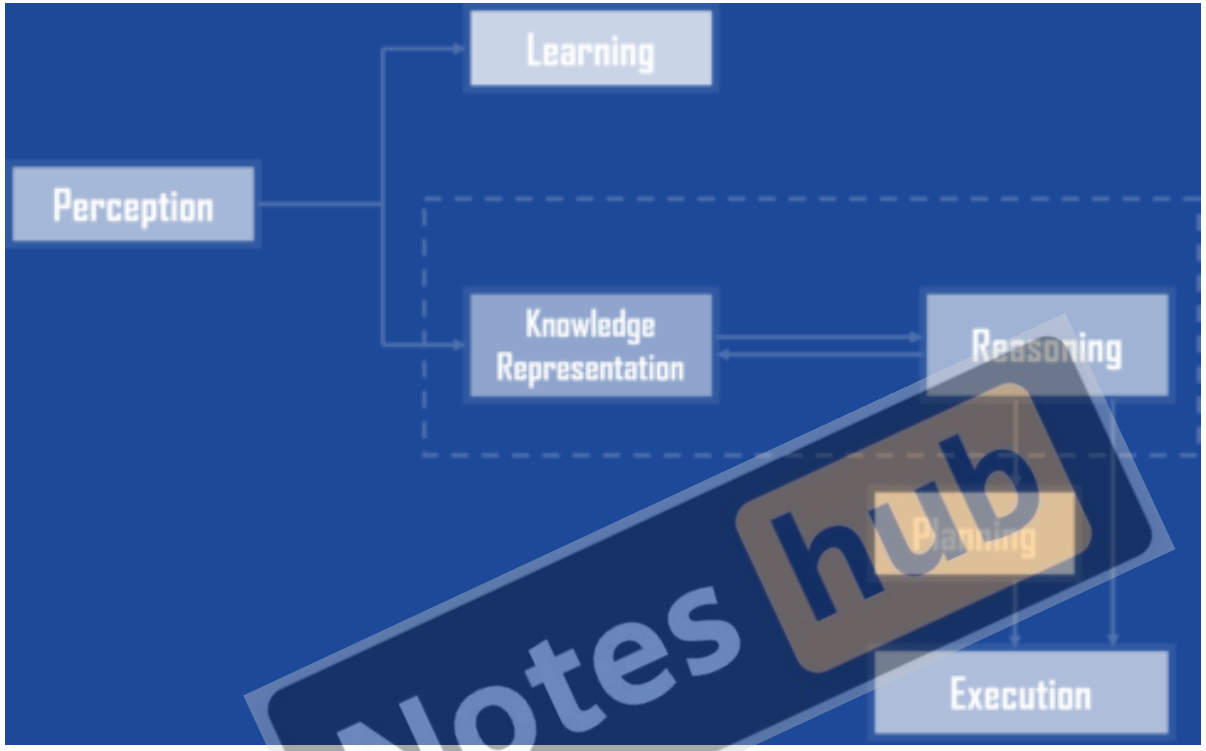
1. **Representation Adequacy:** the ability to represent all of kinds of knowledge that are needed in that domain.
2. **Inferential Adequacy:** the ability to manipulate the representational structures in such a way as to derive new structures corresponding to new knowledge inferred from old.
3. **Inferential Efficiency:** the ability to incorporate into the knowledge structure additional information that can be used to focus the attention of the inference mechanisms in the most promising directions.
4. **Acquisitional Efficiency:** the ability to acquire new information easily. The simplest case involves direct insertion, by a person, of new knowledge into the database.

Cycle of Knowledge Representation in AI

Artificial Intelligent Systems usually consist of various components to display their intelligent behavior. Some of these components include:

- **Perception**
- **Learning**
- **Knowledge Representation & Reasoning**
- **Planning**
- **Execution**

Here is an example to show the different components of the system and how it works:



- The **Perception component** retrieves data or information from the environment. with the help of this component, you can retrieve data from the environment, find out the source of noises and check if the AI was damaged by anything. Also, it defines how to respond when any sense has been detected.

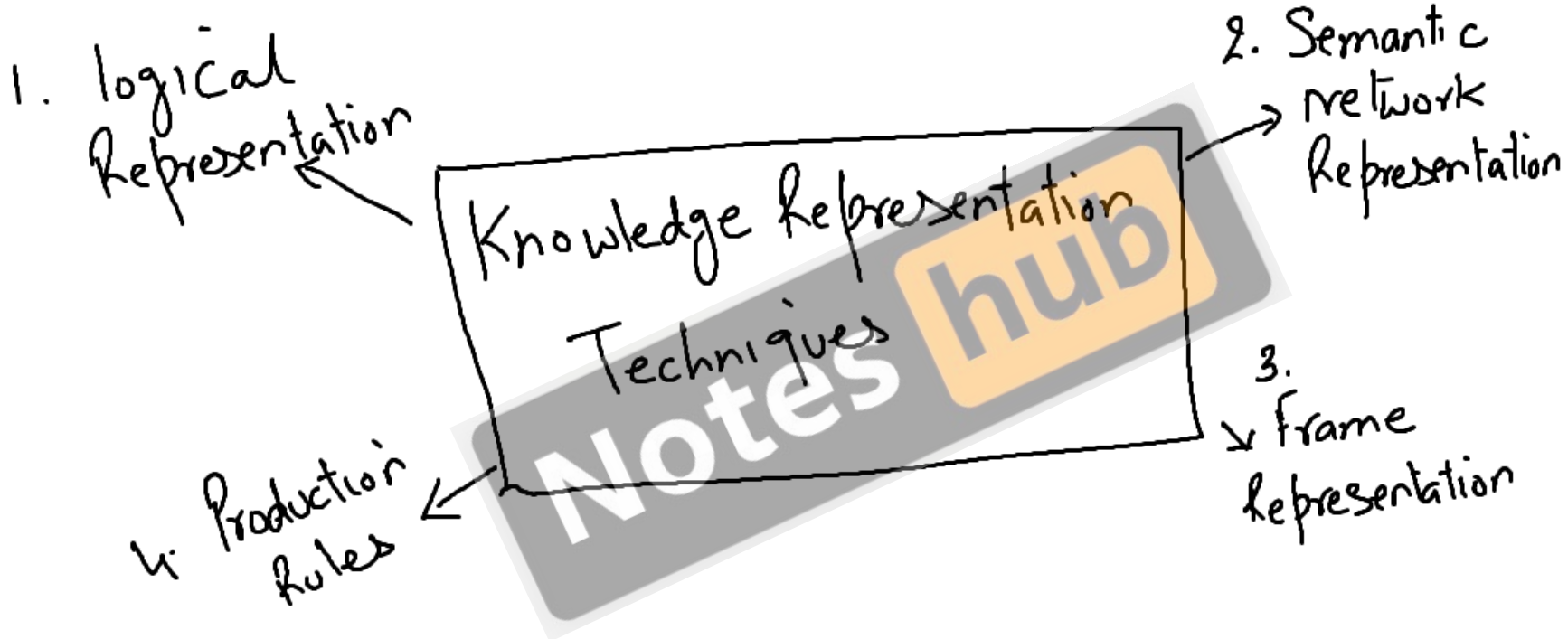
- Then, there is the **Learning Component** that learns from the captured data by the perception component. The goal is to build computers that can be taught instead of programming them. Learning focuses on the process of self-improvement. In order to learn new things, the system requires knowledge acquisition, inference, acquisition of heuristics, faster searches, etc.

- The main component in the cycle is **Knowledge Representation and Reasoning** which shows the human-like intelligence in the machines. Knowledge representation is all about understanding intelligence. Instead of trying to understand or build brains from the bottom up, its goal is to understand and build intelligent behavior from the top-down and focus on what an agent needs to know in order to behave intelligently.

- The **Planning and Execution** components depend on the analysis of knowledge representation and reasoning. Here, planning includes giving an initial state, finding their preconditions and effects, and a sequence of actions to achieve a state in which a particular goal holds. Now once the planning is completed, the final stage is the execution of the entire process.

Relation between Knowledge & Intelligence

Techniques of Knowledge Representation in AI



Logical Representation

Logical representation is a language with some **definite rules** which deal with propositions and has no ambiguity in representation. It represents a conclusion based on various conditions and lays down some important **communication rules**. Also, it consists of precisely defined syntax and semantics which supports the sound inference. Each sentence can be translated into logics using syntax and semantics.

Syntax

- It decides how we can construct legal sentences in logic.
- It determines which symbol we can use in knowledge representation.
- Also, how to write those symbols.

Semantics

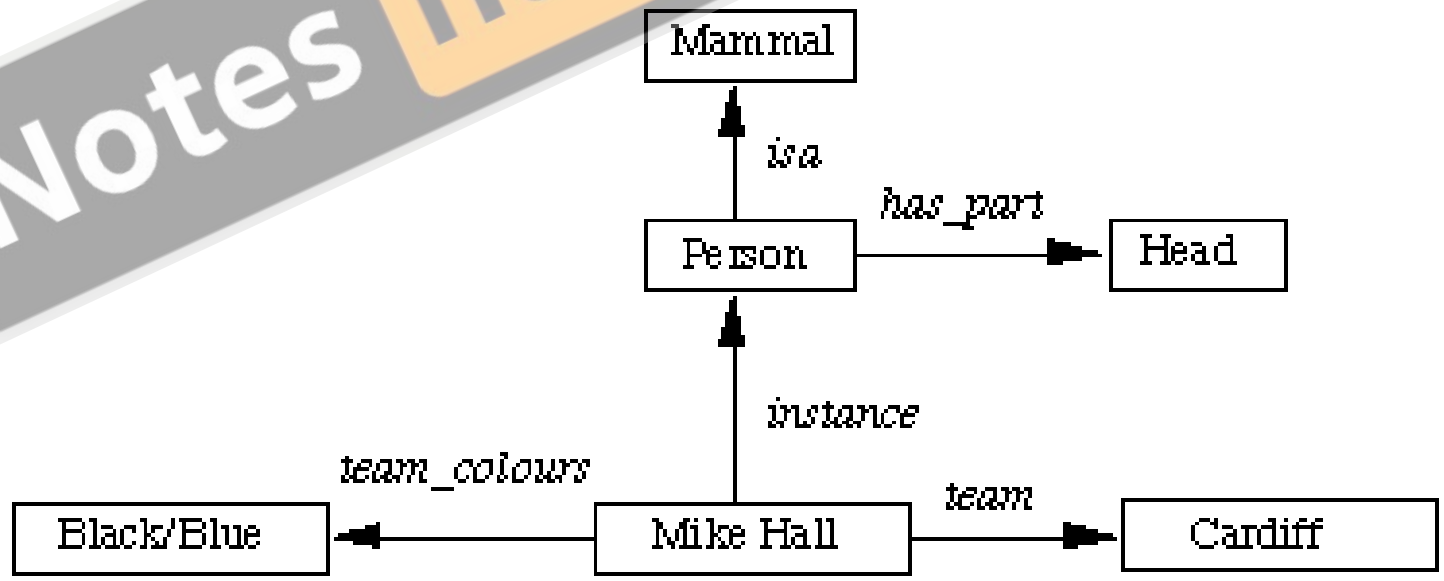
- Semantics are the rules by which we can interpret the sentence in the logic.
- It assigns a meaning to each sentence.

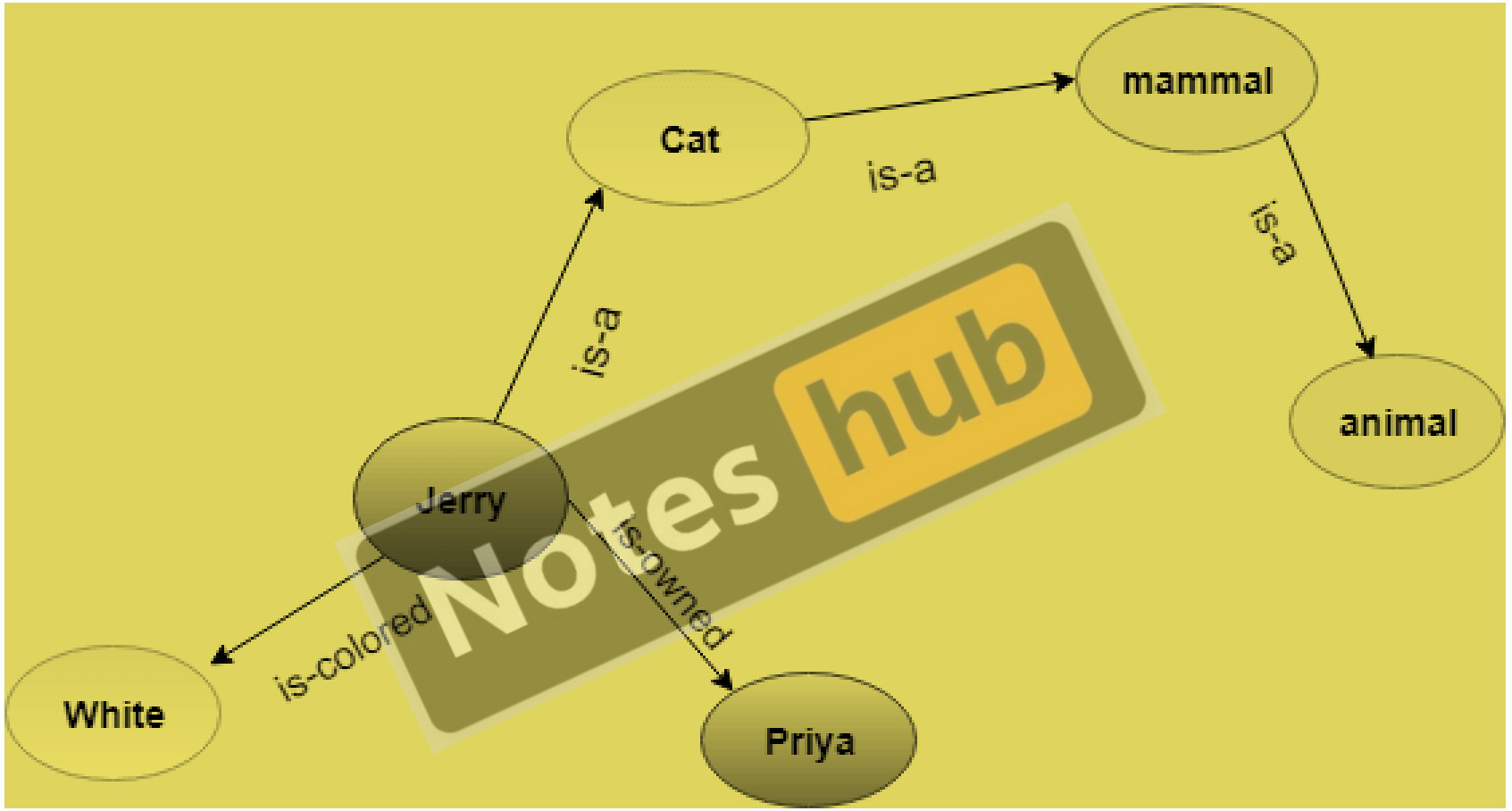
Semantic Network Representation

Semantic networks work as an **alternative** of **predicate logic** for knowledge representation. In Semantic networks, you can represent your knowledge in the form of graphical networks. This network consists of nodes representing objects and arcs which describe the relationship between those objects. Also, it categorizes the object in different forms and links those objects.

This representation consist of two types of relations:

- **IS-A relation (Inheritance)**
- **Kind-of-relation**




Cat
mammal
is-a
is-a
is-a
animal
Jerry
is-colored
is-owned
White
Priya
Notes hub

Frame Representation

A frame is a **record** like structure that consists of a **collection of attributes** and values to describe an entity in the world. These are the AI data structure that divides knowledge into substructures by representing stereotypes situations. Basically, it consists of a collection of slots and slot values of any type and size. Slots have names and values which are called facets.

Production Rules

In production rules, agent checks for the **condition** and if the condition exists then production rule fires and corresponding action is carried out. The condition part of the rule determines which rule may be applied to a problem. Whereas, the action part carries out the associated problem-solving steps. This complete process is called a recognize-act cycle.

Knowledge Representation in AI

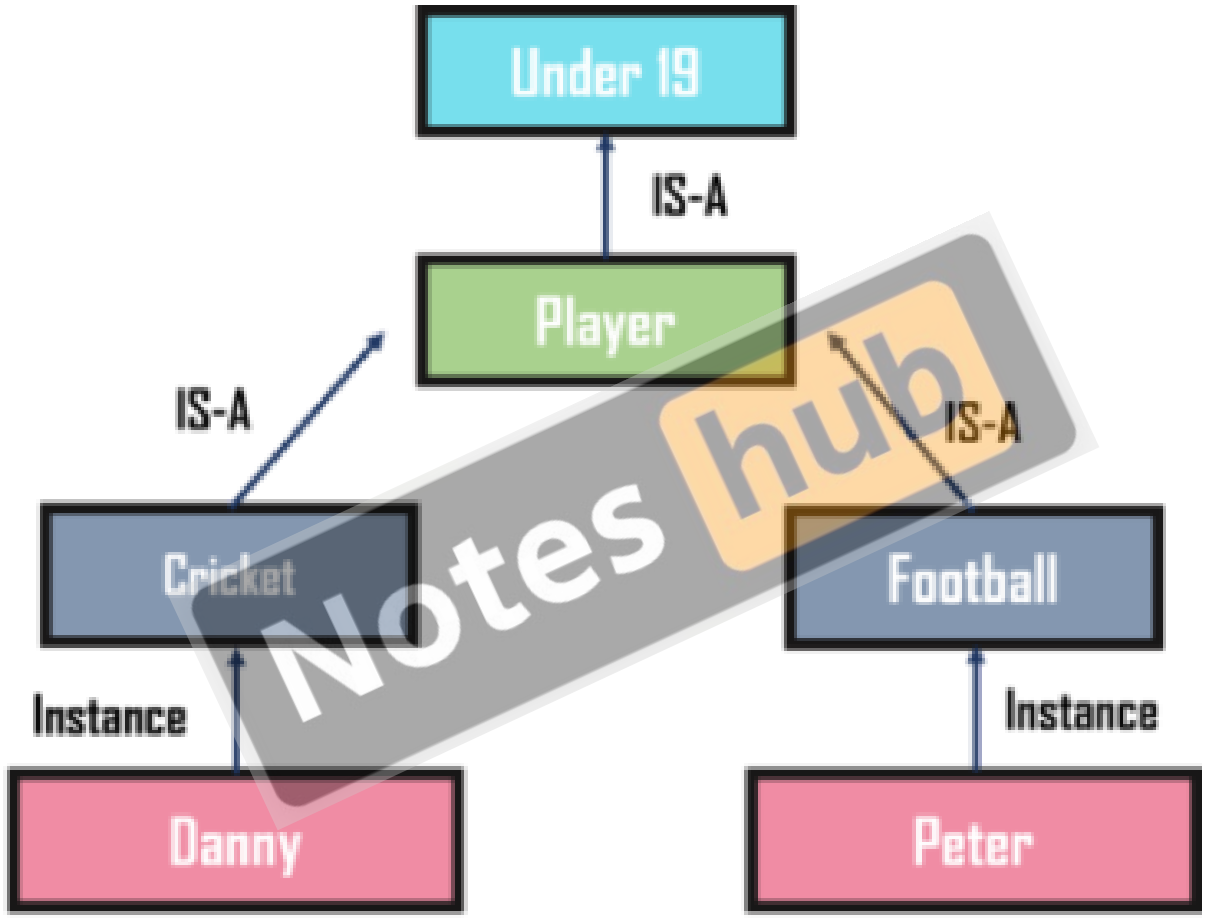
1. Simple Relational Knowledge

It is the simplest way of **storing facts** which uses the relational method. Here, all the facts about a set of the object are set out systematically in columns. Also, this approach of knowledge representation is famous in **database systems** where the relationship between different entities is represented.

Name	Age	Emp ID
John	25	100071
Amanda	23	100056
Sam	27	100042

2. Inheritable Knowledge

In the inheritable knowledge approach, all data must be stored into a **hierarchy of classes** and should be arranged in a generalized form or a hierarchal manner. Also, this approach contains inheritable knowledge which shows a relation between instance and class, and it is called instance relation. In this approach, objects and values are represented in Boxed nodes.


Under 19
IS-A
Player
IS-A
IS-A
Cricket
Football
Instance
Instance
Danny
Peter

3. Inferential Knowledge

The inferential knowledge approach represents **knowledge** in the form of **formal logic**. Thus, it can be used to derive more facts. Also, it guarantees correctness.

Example:

Statement 1: John is a cricketer.

Statement 2: All cricketers are athletes.

Then it can be represented as;

Cricketer(John)

∀x = Cricketer (x) —>Athelete (x)s