Module-II

Knowledge Representation & Reasoning

Knowledge:

Knowledge is the collection of skills and information a person has acquired through experience.

Reasoning:

Reasoning is defined as logical or sensible thinking.

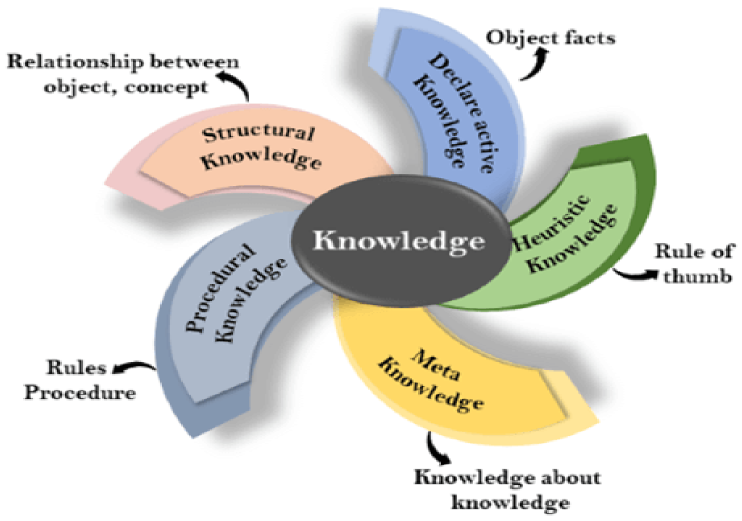
Intelligence:

Intelligence is the ability to apply knowledge.

Knowledge-Based Agent

Knowledge is awareness or familiarity gained by experiences of facts, data, and situations.

Following are the types of knowledge in artificial intelligence:



Types of knowledge

Declarative knowledge(what):

- Declarative knowledge is to know about something.
- It includes concepts, facts, and objects.
- It is also called descriptive knowledge and expressed in declarative sentences.
- It is simpler than procedural language.

Procedural Knowledge(how):

- It is also known as imperative knowledge.

- Procedural knowledge is a type of knowledge which is responsible for knowing **how to do something**.
- It can be directly applied to any task.
- It includes rules, strategies, procedures, agendas, etc.
- Procedural knowledge depends on the task on which it can be applied.

Meta-knowledge:

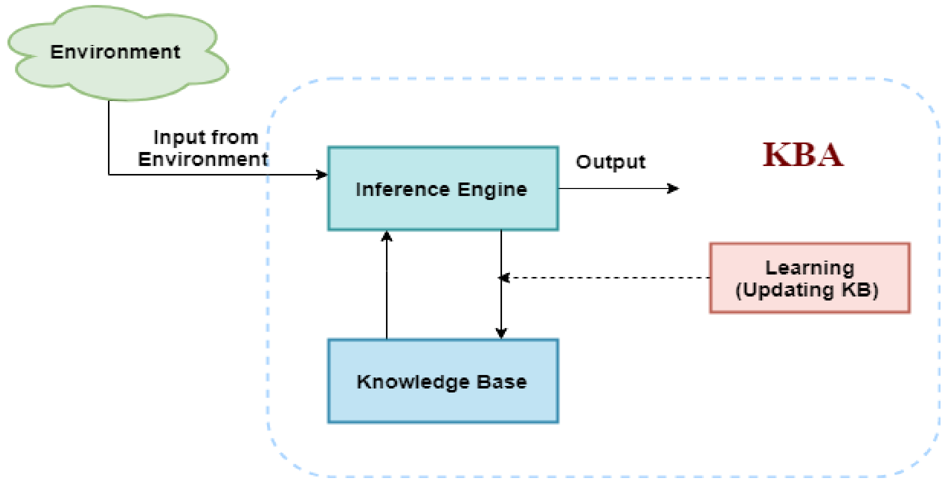
- Knowledge about the other types of knowledge is called Meta-knowledge.
- Used to pick other knowledge that is best suited for solving a problem

Heuristic knowledge:

- Heuristic knowledge is representing knowledge of some experts in a filed or subject.
- Heuristic knowledge is rules of thumb based on previous experiences, awareness of approaches, and which are good to work but not guaranteed.

Structural knowledge:

- Structural knowledge is basic knowledge to problem-solving.
- It describes relationships between various concepts such as kind of, part of, and grouping of something.
- It describes the relationship that exists between concepts or objects.
- Knowledge-Based Agent
- An intelligent agent needs **knowledge** about the real world for taking decisions and **reasoning** to act efficiently.
- Knowledge-based agents are those agents who have the capability of **maintaining an internal state of knowledge, reason over that knowledge, update their knowledge after observations and take actions. These agents can represent the world with some formal representation and act intelligently**.
- Knowledge-based agents are composed of two main parts:
    - **Knowledge-base**
    - **Inference system**



A knowledge-based agent must able to do the following:

- An agent should be able to represent states, actions, etc.
- An agent Should be able to incorporate new percepts
- An agent can update the internal representation of the world
- An agent can deduce the internal representation of the world

An agent can deduce appropriate actions.

<u>Knowledge base:</u>

- Knowledge-base is a central component of a knowledge-based agent, it is also known as KB.
- It is a collection of sentences (here 'sentence' is a technical term and it is not identical to sentence in English).
- These sentences are expressed in a language which is called a knowledge representation language.
- The Knowledge-base of KBA stores fact about the world.

<u>Why use a knowledge base?</u>

- Knowledge-base is required for updating knowledge for an agent to learn with experiences and take action as per the knowledge.

Inference system

- Inference means deriving new sentences from old. Inference system allows us to add a new sentence to the knowledge base. A sentence is a proposition about the world. Inference system applies logical rules to the KB to deduce new information.
- Inference system generates new facts so that an agent can update the KB. An inference system works mainly in two rules which are given as:
- **Forward chaining**

- **Backward chaining**

Approaches to designing a knowledge-based agent:

There are mainly two approaches to build a knowledge-based agent:

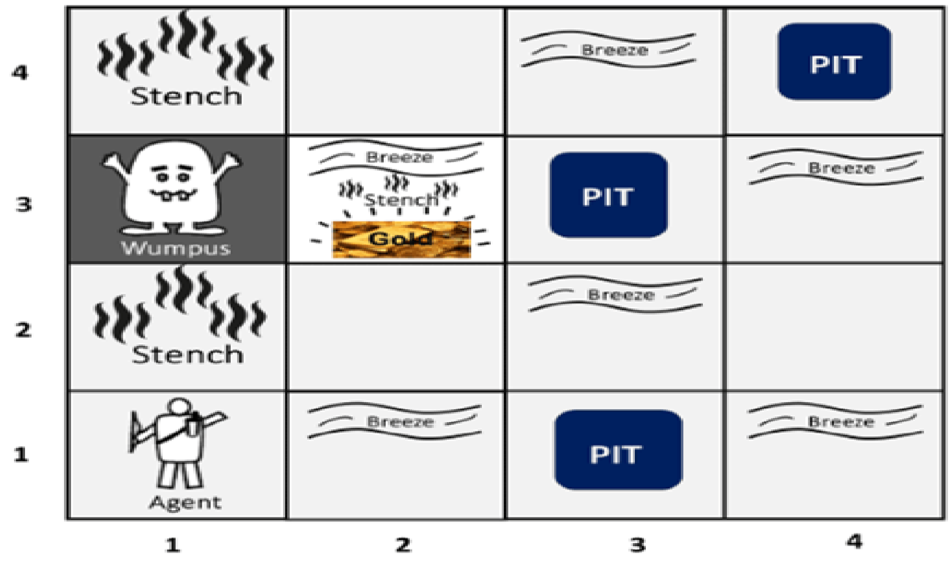
1. **Declarative approach:** We can create a knowledge-based agent by initializing with an empty knowledge base and telling the agent all the sentences with which we want to start with. This approach is called Declarative approach.

2. Procedural approach: In the procedural approach, we directly encode desired behavior as a program code. Which means we just need to write a program that already encodes the desired behavior or agent.

Wumpus world:

- The Wumpus world is a cave which has 4/4 rooms connected with passageways.
- The cave has a room with a beast which is called Wumpus, who eats anyone who enters the room.
- The Wumpus can be shot by the agent, but the agent has a single arrow.
- In the Wumpus world, there are some Pits rooms which are bottomless, and if agent falls in Pits, then he will be stuck there forever.
- The exciting thing with this cave is that in one room there is a possibility of finding a heap of gold.
- So the agent goal is to find the gold and climb out the cave without fallen into Pits or eaten by Wumpus.

The agent will get a reward if he comes out with gold, and he will get a penalty if eaten by Wumpus or falls in the pit.



There are also some components which can help the agent to navigate the cave. These components are given as follows:

- The rooms adjacent to the Wumpus room are smelly, so that it would have some stench.

- The room adjacent to PITs has a breeze, so if the agent reaches near to PIT, then he will perceive the breeze.
- There will be glitter in the room if and only if the room has gold.
- The Wumpus can be killed by the agent if the agent is facing to it, and Wumpus will emit a horrible scream which can be heard anywhere in the cave.

PEAS description of Wumpus world:

To explain the Wumpus world we have given PEAS description as below:

Performance measure:

- +1000 reward points if the agent comes out of the cave with the gold.
- -1000 points penalty for being eaten by the Wumpus or falling into the pit.
- -1 for each action, and -10 for using an arrow.
- The game ends if either agent dies or came out of the cave.

Environment:

- A 4*4 grid of rooms.
- The agent initially in room square [1, 1], facing toward the right.
- Location of Wumpus and gold are chosen randomly except the first square [1,1].
- Each square of the cave can be a pit with probability 0.2 except the first square.

Actuators:

- Left turn,
- Right turn
- Move forward
- Grab
- Release
- Shoot.

Sensors:

- The agent will perceive the **stench** if he is in the room adjacent to the Wumpus. (Not diagonally).
- The agent will perceive **breeze** if he is in the room directly adjacent to the Pit.
- The agent will perceive the **glitter** in the room where the gold is present.
- The agent will perceive the **bump** if he walks into a wall.

- When the Wumpus is shot, it emits a horrible **scream** which can be perceived anywhere in the cave.
- These percepts can be represented as five element list, in which we will have different indicators for each sensor.
- Example if agent perceives stench, breeze, but no glitter, no bump, and no scream then it can be represented as: **[Stench, Breeze, None, None, None]**.

Exploring the Wumpus world:

Now we will explore the Wumpus world and will determine how the agent will find its goal by applying logical reasoning.

Agent's First step:

- Initially, the agent is in the first room or on the square [1,1], and we already know that this room is safe for the agent, so to represent on the below diagram (a) that room is safe we will add symbol OK.
- Symbol A is used to represent agent, symbol B for the breeze, G for Glitter or gold, V for the visited room, P for pits, W for Wumpus.
- At Room [1,1] agent does not feel any breeze or any Stench which means the adjacent squares are also OK.

Agent's second Step:

- Now agent needs to move forward, so it will either move to [1, 2], or [2,1].
- Let's suppose agent moves to the room [2, 1], at this room agent perceives some breeze which means Pit is around this room.
- The pit can be in [3, 1], or [2,2], so we will add symbol P? to say that, is this Pit room?
- Now agent will stop and think and will not make any harmful move.
- The agent will go back to the [1, 1] room.
- The room [1,1], and [2,1] are visited by the agent, so we will use symbol V to represent the visited squares.

Agent's third step:

- At the third step, now agent will move to the room [1,2] which is OK.
- In the room [1,2] agent perceives a stench which means there must be a Wumpus nearby.
- But Wumpus cannot be in the room [1,1] as by rules of the game, and also not in [2,2] (Agent had not detected any stench when he was at [2,1]).
- Therefore agent infers that Wumpus is in the room [1,3], and in current state, there is no breeze which means in [2,2] there is no Pit and no Wumpus.
- So it is safe, and we will mark it OK, and the agent moves further in [2,2].

Agent's fourth step:

- At room [2,2], here no stench and no breezes present so let's suppose agent decides to move to [2,3].
- At room [2,3] agent perceives glitter, so it should grab the gold and climb out of the cave.

Logic:

The philosophical definition:

Logic is a description of how one should think.

In the context of AI:

Logic is "formal," which means it resembles math in its clarity and lack of ambiguity.

- Logic
  - Propositional logic(true or false)
  - Predicate logic(quantifiers)
- Rules(if-then)
- Semantic net(Google graph):meaningful graph
- Frame(slot and filler):object and attribute
- Script

Propositional logic:

- Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions.
- A proposition is a declarative statement which is either true or false.
- It is a technique of knowledge representation in logical and mathematical form.

Example:

a) It is Sunday.

b) The Sun rises from West (False proposition)

c) 3+3= 7(False proposition)

d) 5 is a prime number.

Basic facts about propositional logic:

- 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.
- Propositional logic consists of an object, relations or function, and **logical connectives**.
- These connectives are also called logical operators.
- The propositions and connectives are the basic elements of the propositional logic.
- Connectives can be said as a logical operator which connects two sentences.
- A proposition formula which is always true is called **tautology**, and it is also called a valid sentence.
- A proposition formula which is always false is called **Contradiction**.
- A proposition formula which has both true and false values is called
- Statements which are questions, commands, or opinions are not propositions such as "**Where is Rohini**", "**How are you**", "**What is your name**", are not propositions.

Syntax of propositional logic:

The syntax of propositional logic defines the allowable sentences for the knowledge representation. There are two types of Propositions:

Atomic Propositions

Compound propositions

Atomic Proposition:

Atomic propositions are the simple propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false.

Example:

a) 2+2 is 4, it is an atomic proposition as it is a **true** fact.

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

Compound proposition:

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

Example:

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

b) "Ankit is a doctor, and his clinic is in Mumbai."

Logical Connectives:

Negation:

A sentence such as ¬ P is called negation of P.

A literal can be either Positive literal or negative literal.

Conjunction:

A sentence which has ∧ connective such as, **P ∧ Q** is called a conjunction.
Example:

Rohan is intelligent and hardworking. It can be written as,
P= Rohan is intelligent,
Q= Rohan is hardworking. → P∧ Q.

Disjunction:

A sentence which has ∨ connective, such as **P ∨ Q**. is called disjunction, where P and Q are the propositions.

Example:

"Ritika is a doctor or Engineer",
Here P= Ritika is Doctor. Q= Ritika is Doctor, so we can write it as **P ∨ Q**.

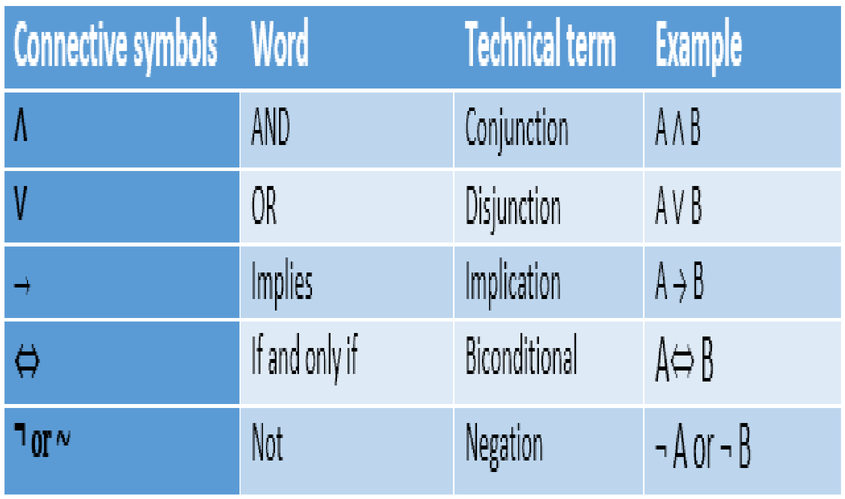
Implication:

A sentence such as P → Q, is called an implication. Implications are also known as if-then rules. It can be represented as
If it is raining, then the street is wet.
Let P=It is raining, and Q=Street is wet, so it is represented as P → Q

Biconditional: A sentence such as **P⇔ Q is a Biconditional sentence,**

Example

If I am breathing, then I am alive
P= I am breathing, Q= I am alive, it can be represented as P ⇔ Q.



Connective symbols	Word	Technical term	Example
∧	AND	Conjunction	A ∧ B
∨	OR	Disjunction	A ∨ B
→	Implies	Implication	A → B
⇔	If and only if	Biconditional	A⇔ B
¬ or ~	Not	Negation	¬A or ¬B

For Negation:

P	¬ P
True	**False**
False	**True**

For Conjunction:

P	Q	P∧ Q
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**False**

For disjunction:

P	Q	P ∨ Q.
True	True	**True**
False	True	**True**
True	False	**True**
False	False	**False**

For Implication:

P	Q	P→ Q
True	True	**True**
True	False	**False**
False	True	**True**
False	False	**True**

For Biconditional:

P	Q	P⇔ Q
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**True**

P	Q	R	¬R	Pv Q	PvQ→¬R
True	True	True	False	True	False
True	True	False	True	True	True
True	False	True	False	True	False
True	False	False	True	True	True
False	True	True	False	True	False
False	True	False	True	True	True
False	False	True	False	False	True
False	False	False	True	False	True

Properties of Operators:

Commutativity:

P∧ Q= Q ∧ P, or

P ∨ Q = Q ∨ P.

Associativity:

$(P \wedge Q) \wedge R = P \wedge (Q \wedge R)$,

$(P \vee Q) \vee R = P \vee (Q \vee R)$

Identity element:

$P \wedge True = P$,

$P \vee True = True$.

Distributive:

$P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$.

$P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$.

DE Morgan's Law:

$\neg (P \wedge Q) = (\neg P) \vee (\neg Q)$

$\neg (P \vee Q) = (\neg P) \wedge (\neg Q)$.

Double-negation elimination:

$\neg (\neg P) = P$.

Limitations of Propositional logic:

We cannot represent relations like ALL, some, or none with propositional logic.

Example:

All the girls are intelligent.

Some apples are sweet.

Propositional logic has limited expressive power.

In propositional logic, we cannot describe statements in terms of their properties or logical relationships.

First-Order logic:

Consider the following sentence, which we cannot represent using PL logic.

"Some humans are intelligent", or

"Sachin likes cricket."

To represent the above statements, PL logic is not sufficient, so we required some more powerful logic, such as first-order logic.

First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.

FOL is sufficiently expressive to represent the natural language statements in a concise way.

First-order logic is also known as **Predicate logic or First-order predicate logic**. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.

First-order logic (like natural language) does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:

Objects: A, B, people, numbers, colors, wars, theories, squares, pits, wumpus, ...

Relations: It can be unary relation such as: red, round, is adjacent, **or n-any relation such as:** the sister of, brother of, has color, comes between

Function: Father of, best friend, third inning of, end of, ...

As a natural language, first-order logic also has two main parts:

Syntax

Semantics

Syntax of First-Order logic:

constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf, ...
Connectives	∧, ∨, ¬, ⇒, ⇔
Equality	==
Quantifier	∀, ∃

Atomic sentences:

Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.

We can represent atomic sentences as **Predicate (term1, term2, ..., term n)**.

Example: Ravi and Ajay are brothers: => Brothers(Ravi, Ajay).
Chinky is a cat: => cat (Chinky).

Complex Sentences:

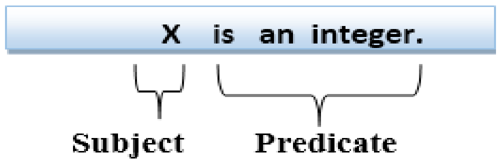
Complex sentences are made by combining atomic sentences using connectives.

First-order logic statements can be divided into two parts:

Subject: Subject is the main part of the statement.

Predicate: A predicate can be defined as a relation, which binds two atoms together in a statement.

Consider the statement: "x is an integer.", it consists of two parts, the first part x is the subject of the statement and second part "is an integer," is known as a predicate.



Quantifiers in First-order logic:

A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.

These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifier:

Universal Quantifier, (for all, everyone, everything)

Existential quantifier, (for some, at least one).

Universal Quantifier:

Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing.

The Universal quantifier is represented by a symbol ∀, which resembles an inverted A.

If x is a variable, then ∀x is read as:

For all x

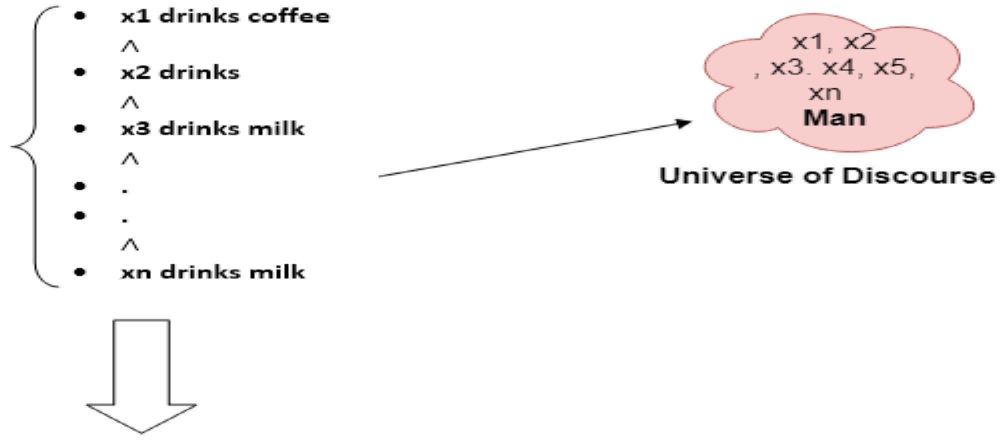
For each x

For every x.

Example:

All man drink coffee.

Let a variable x which refers to a cat so all x can be represented in UOD as below:



So in shorthand notation, we can write it as :

∀x man(x) → drink (x, coffee).

It will be read as: There are all x where x is a man who drink coffee.

Existential Quantifier:

Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something.

It is denoted by the logical operator ∃, which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

If x is a variable, then existential quantifier will be ∃x or ∃(x). And it will be read as:

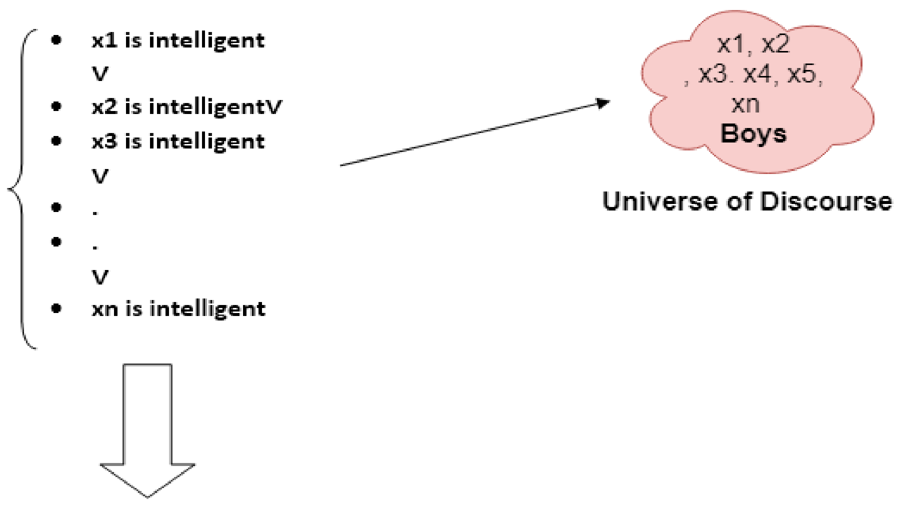
There exists a 'x.'

For some 'x.'

For at least one 'x.'

Example:

Some boys are intelligent.



So in short-hand notation, we can write it as:

∃x: boys(x) ∧ intelligent(x)

It will be read as: There are some x where x is a boy who is intelligent.

Points to remember:

The main connective for universal quantifier **∀** is implication →.

The main connective for existential quantifier **∃** is and **∧**.

Properties of Quantifiers:

In universal quantifier, ∀x∀y is similar to ∀y∀x.

In Existential quantifier, ∃x∃y is similar to ∃y∃x.

∃x∀y is not similar to ∀y∃x.

1. All birds fly.
In this question the predicate is "**fly(bird)**."
And since there are all birds who fly so it will be represented as follows.
∀x bird(x) →fly(x).

2. Every man respects his parent.
In this question, the predicate is "**respect(x, y)," where x=man, and y= parent**.
Since there is every man so will use ∀, and it will be represented as follows:
∀x man(x) → respects (x, parent).

3. Some boys play cricket.
In this question, the predicate is "**play(x, y)**," where x= boys, and y= game. Since there are some boys so we will use **∃, and it will be represented as**:

∃x boys(x) → play(x, cricket).

4. Not all students like both Mathematics and Science.
In this question, the predicate is "**like(x, y)," where x= student, and y= subject**.
Since there are not all students, so we will use **∀ with negation, so** following representation for this:
¬∀ (x) [student(x) → like(x, Mathematics) ∧ like(x, Science)].

5. Only one student failed in Mathematics.
In this question, the predicate is "**failed(x, y)," where x= student, and y= subject**.
Since there is only one student who failed in Mathematics, so we will use following representation for this:
∃(x) [student(x) → failed (x, Mathematics) ∧∀ (y) [¬(x==y) ∧ student(y) → ¬failed (x, Mathematics)].

Free and Bound Variables:

The quantifiers interact with variables which appear in a suitable way. There are two types of variables in First-order logic which are given below:

Free Variable: A variable is said to be a free variable in a formula if it occurs outside the scope of the quantifier.

Example: ∀x ∃(y)[P (x, y, z)], where z is a free variable.

Bound Variable: A variable is said to be a bound variable in a formula if it occurs within the scope of the quantifier.

Example: ∀x [A (x) B(y)], here x and y are the bound variables.

Inference in First-Order Logic

Inference in First-Order Logic is used to deduce new facts or sentences from existing sentences. Before understanding the FOL inference rule, let's understand some basic terminologies used in FOL.

Substitution:

Substitution is a fundamental operation performed on terms and formulas. It occurs in all inference systems in first-order logic. The substitution is complex in the presence of quantifiers in FOL. If we write **F[a/x]**, so it refers to substitute a constant "**a**" in place of variable "**x**".

Equality:

First-Order logic does not only use predicate and terms for making atomic sentences but also uses another way, which is equality in FOL. For this, we can use **equality symbols** which specify that the two terms refer to the same object.

Example: Brother (John) = Smith.

As in the above example, the object referred by the **Brother (John)** is similar to the object referred by **Smith**. The equality symbol can also be used with negation to represent that two terms are not the same objects.

Example: ¬(x=y) which is equivalent to x ≠y.

FOL inference rules for quantifier:

As propositional logic we also have inference rules in first-order logic, so following are some basic inference rules in FOL:

Universal Generalization

Universal Instantiation

Existential Instantiation

Existential introduction

Universal Generalization:

Universal generalization is a valid inference rule which states that if premise P(c) is true for any arbitrary element c in the universe of discourse, then we can have a conclusion as ∀ x P(x).

It can be represented as: .

This rule can be used if we want to show that every element has a similar property.

In this rule, x must not appear as a free variable.

Example: Let's represent, P(c): "**A byte contains 8 bits**", so for **∀ x P(x)** "**All bytes contain 8 bits**.", it will also be true.

Universal Instantiation:

Universal instantiation is also called as universal elimination or UI is a valid inference rule. It can be applied multiple times to add new sentences.

The new KB is logically equivalent to the previous KB.

As per UI, **we can infer any sentence obtained by substituting a ground term for the variable**.

The UI rule state that we can infer any sentence P(c) by substituting a ground term c (a constant within domain x) from **∀ x P(x) for any object in the universe of discourse**.

It can be represented as: Universal instantiation is also called as universal elimination or UI is a valid inference rule. It can be applied multiple times to add new sentences.

The new KB is logically equivalent to the previous KB.

As per UI, **we can infer any sentence obtained by substituting a ground term for the variable**.

The UI rule state that we can infer any sentence P(c) by substituting a ground term c (a constant within domain x) from **∀ x P(x) for any object in the universe of discourse**.

It can be represented as:

$$\frac{\forall x\, P(x)}{P(c)}$$

Example:1.

IF "Every person like ice-cream"=> ∀x P(x) so we can infer that
"John likes ice-cream" => P(c)

Example: 2.

Let's take a famous example,

"All kings who are greedy are Evil." So let our knowledge base contains this detail as in the form of FOL:

∀x king(x) ∧ greedy (x) → Evil (x),

So from this information, we can infer any of the following statements using Universal Instantiation:

King(John) ∧ Greedy (John) → Evil (John),

King(Richard) ∧ Greedy (Richard) → Evil (Richard),

King(Father(John)) ∧ Greedy (Father(John)) → Evil (Father(John)),

Existential Instantiation:

Existential instantiation is also called as Existential Elimination, which is a valid inference rule in first-order logic.

It can be applied only once to replace the existential sentence.

The new KB is not logically equivalent to old KB, but it will be satisfiable if old KB was satisfiable.

This rule states that one can infer P(c) from the formula given in the form of ∃x P(x) for a new constant symbol c.

The restriction with this rule is that c used in the rule must be a new term for which P(c) is true.

It can be represented as:

$$\frac{\exists x\ P(x)}{P(c)}$$

Example:

From the given sentence: **∃x Crown(x) ∧ OnHead(x, John),**

So we can infer: **Crown(K) ∧ OnHead(K, John),** as long as K does not appear in the knowledge base.

The above used K is a constant symbol, which is called **Skolem constant**.

The Existential instantiation is a special case of **Skolemization process**.

Existential introduction

An existential introduction is also known as an existential generalization, which is a valid inference rule in first-order logic.

This rule states that if there is some element c in the universe of discourse which has a property P, then we can infer that there exists something in the universe which has the property P.

It can be represented as:

Example: Let's say that,
"Priyanka got good marks in English."
"Therefore, someone got good marks in English."

Generalized Modus Ponens Rule:

For the inference process in FOL, we have a single inference rule which is called Generalized Modus Ponens. It is lifted version of Modus ponens.

Generalized Modus Ponens can be summarized as, " P implies Q and P is asserted to be true, therefore Q must be True."

According to Modus Ponens, for atomic sentences **pi, pi', q**. Where there is a substitution θ such that SUBST **(θ, pi',) = SUBST(θ, pi)**, it can be represented as:

$$\frac{p1', p2', \ldots, pn', (p1 \land p2 \land \ldots \land pn \Rightarrow q)}{SUBST(\theta, q)}$$

Example:

We will use this rule for Kings are evil, so we will find some x such that x is king, and x is greedy so we can infer that x is evil.

Here let say, p1' is king(John)   p1 is king(x)

p2' is Greedy(y)   p2 is Greedy(x)

θ is {x/John, y/John}   q is evil(x)

SUBST(θ,q).

Inference engine:

The inference engine is the component of the intelligent system in artificial intelligence, which applies logical rules to the knowledge base to infer new information from known facts. The first inference engine was part of the expert system. Inference engine commonly proceeds in two modes, which are:

Forward chaining

Backward chaining

Horn Clause and Definite clause:

Horn clause and definite clause are the forms of sentences, which enables knowledge base to use a more restricted and efficient inference algorithm. Logical inference algorithms use forward and backward chaining approaches, which require KB in the form of the **first-order definite clause**.

Definite clause: A clause which is a disjunction of literals with **exactly one positive literal** is known as a definite clause or strict horn clause.

Horn clause: A clause which is a disjunction of literals with **at most one positive literal** is known as horn clause. Hence all the definite clauses are horn clauses.

Example: (¬ p V ¬ q V k). It has only one positive literal k.

It is equivalent to p ∧ q → k.

Forward Chaining

Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine. Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.

The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.

Properties of Forward-Chaining:

It is a down-up approach, as it moves from bottom to top.

It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.

Forward-chaining approach is also called as data-driven as we reach to the goal using available data.

Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

Example:

"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."

Prove that **"Robert is criminal."**

To solve the above problem, first, we will convert all the above facts into first-order definite clauses, and then we will use a forward-chaining algorithm to reach the goal.

Facts Conversion into FOL:

It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)
American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)

Country A has some missiles. **?p Owns(A, p) ∧ Missile(p)**. It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.
Owns(A, T1) ......(2)
Missile(T1) ....(3)

All of the missiles were sold to country A by Robert.
?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) ......(4)

Missiles are weapons.
Missile(p) → Weapons (p) ....(5)

Enemy of America is known as hostile.
Enemy(p, America) →Hostile(p) ....(6)

Country A is an enemy of America.
Enemy (A, America) ....(7)

Robert is American
American(Robert). ....(8)

Backward Chaining:

Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine. A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.

Properties of backward chaining:

It is known as a top-down approach.

Backward-chaining is based on modus ponens inference rule.

In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.

It is called a goal-driven approach, as a list of goals decides which rules are selected and used.

Backward -chaining algorithm is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.

The backward-chaining method mostly used a **depth-first search** strategy for proof.

Example:

In backward-chaining, we will use the same above example, and will rewrite all the rules.

American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)
Owns(A, T1) ....(2)

Missile(T1)

?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) ......(4)

Missile(p) → Weapons (p) ....(5)

Enemy(p, America) →Hostile(p) ....(6)

Enemy (A, America) ....(7)

American(Robert). ....(8)

Difference between backward chaining and forward chaining

S. No.	Forward Chaining	Backward Chaining
1.	Forward chaining starts from known facts and applies inference rule to extract more data unit it reaches to the goal.	Backward chaining starts from the goal and works backward through inference rules to find the required facts that support the goal.
2.	It is a bottom-up approach	It is a top-down approach
3.	Forward chaining is known as data-driven inference technique as we reach to the goal using the available data.	Backward chaining is known as goal-driven technique as we start from the goal and divide into sub-goal to extract the facts.

4.	Forward chaining reasoning applies a breadth-first search strategy.	Backward chaining reasoning applies a depth-first search strategy.
5.	Forward chaining tests for all the available rules	Backward chaining only tests for few required rules.
6.	Forward chaining is suitable for the planning, monitoring, control, and interpretation application.	Backward chaining is suitable for diagnostic, prescription, and debugging application.
7.	Forward chaining can generate an infinite number of possible conclusions.	Backward chaining generates a finite number of possible conclusions.
8.	It operates in the forward direction.	It operates in the backward direction.
9.	Forward chaining is aimed for any conclusion.	Backward chaining is only aimed for the required data.