UNIT-4: Knowledge representation (14 Hrs)

Syllabus

Unit IV: Knowledge Representation (14 Hrs.)

4.1. Definition and importance of Knowledge, Issues in Knowledge Representation, Knowledge Representation Systems, Properties of Knowledge Representation Systems.

4.2. Types of Knowledge Representation Systems: Semantic Nets, Frames, Conceptual Dependencies, Scripts, Rule Based Systems, Propositional Logic, Predicate Logic

4.3. Propositional Logic(PL): Syntax, Semantics, Formal logic-connectives, truth tables, tautology, validity, well-formed-formula, Inference using Resolution, Backward Chaining and Forward Chaining

4.4. Predicate Logic: FOPL, Syntax, Semantics, Quantification, Inference with FOPL: By converting into PL (Existential and universal instantiation), Unification and lifting, Inference using resolution

4.5. Handling Uncertain Knowledge, Radom Variables, Prior and Posterior Probability, Inference using Full Joint Distribution, Bayes' Rule and its use, Bayesian Networks, Reasoning in Belief Networks

4.6. Fuzzy Logic

Definition and importance of Knowledge

Knowledge is awareness or familiarity gained by experiences of facts, data, and situations or **it can be defined as** the information about a domain that can be used to solve problems in that domain. To solve many problems requires much knowledge, and this knowledge must be represented in the computer. As part of designing a program to solve problems, we must define how the knowledge will be represented. In case of problem solving by computer, Knowledge is the information about a particular domain that can be used to solve problems in that domain.

Knowledge may be declarative or procedural; Procedural knowledge is compiled knowledge related to the performance of some task. For example, the steps used to solve an algebraic equation. Similarly, Declarative knowledge is passive knowledge expressed as statements of facts about the world. The facts are declarative sentences that are either true or false. For example, personnel data in a database, such data are explicit pieces of independent knowledge.

There are significant differences between knowledge and intelligence. Knowledge is the collection of skills and information a person's acquired through experience. Knowing how to apply that knowledge to problem- solving and decision-making is intelligence.

Knowledge is important in problem solving by computer because to solve a problem by a computer, it first must be represented in computer understandable form. The general framework for solving problems by computer can be given as shown in figure below:

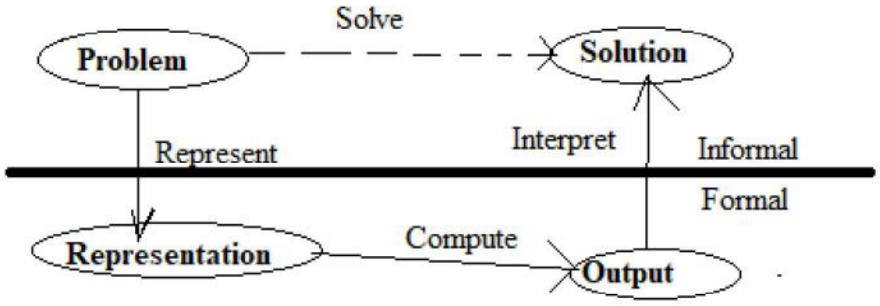


Fig: Knowledge Representation Framework

As shown in above figure,

- To solve problems, Computer requires a well-defined problem description to process and provide well-defined acceptable solution.
- To collect fragments of knowledge we need first to formulate a description in our spoken language and then represent it in formal language so that computer can understand.
- The computer can then use an algorithm to compute an answer.

Hence, to solve problems, it requires full of knowledge about the problem domain, and this knowledge must be represented in the computer. As part of designing a program to solve problems, we must define how the knowledge will be represented in computer understandable form.

The process of converting informal knowledge into computer understandable form is known as knowledge representation scheme. A representation scheme specifies the form of the knowledge. A knowledge base is the representation of all of the knowledge that is stored by an agent.

Following are the types of knowledge in artificial intelligence:

1. Declarative Knowledge:

- 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.

2. Procedural Knowledge

- 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.

3. Meta-knowledge:

- Knowledge about the other types of knowledge is called Meta-knowledge.

4. 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.

5. 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 Representation

Human beings are good at understanding, reasoning and interpreting knowledge. And using this knowledge, they are able to perform various actions in the real world. But how do machines perform the same? Here, **how machines do all these things** comes under knowledge representation and reasoning. Hence, Knowledge representation is the question of **how human knowledge can be encoded into a form that can be handled by computer algorithms** and heuristics.

Basically, **Knowledge Representation** is a study of how the beliefs, intentions, and judgments of an intelligent agent can be expressed suitably for automated reasoning. One of the primary purposes of Knowledge Representation includes modelling intelligent behaviour for an agent.

Knowledge Representation and Reasoning (KR, KRR) represents information from the real world for a computer to understand and then utilize this knowledge to solve complex real-life problems like communicating with human beings in natural language. Knowledge representation in AI is not just about storing data in a database, it allows a machine to learn from that knowledge and behave intelligently like a human being.

The different kinds of knowledge that need to be represented in AI include:

- **Object:** All the facts about objects in our world domain.
- **Events:** Events are the actions which occur in our world.
- **Performance:** It describe behaviour which involves knowledge about how to do things.
- **Meta-knowledge:** It is knowledge about what we know.
- **Facts:** Facts are the truths about the real world and what we represent.
- **Knowledge-Base:** It is the main component of any human, i.e., having a knowledge base. This refers to a group of information regarding any discipline, field, etc. For example, a knowledge-base regarding constructing roads.
- A knowledge base is an easily accessible data storage hub that contains information about a certain product, service, topic, or concept. Artificial knowledge (also called machine

knowledge) is unnatural, intelligent behaviour that machines (computers) generate. Artificial intelligence (AI) actually stores and sorts knowledge in knowledge bases.

Hence, the main objective of knowledge representation is to draw the conclusions from the knowledge, in short, the goal of knowledge Representation is to facilitate inference (conclusions) from knowledge.

Issue in Knowledge Representations

The main objective of knowledge representation is to draw the conclusions from the knowledge, but there are many issues associated with the use of knowledge representation techniques. Some of the issue regarding knowledge representation can be formulated as questions as follows:

- How can the problem be represented?
- What distinctions in the world are needed to solve the problem?
- What specific knowledge about the world is required?
- How can an agent acquire the knowledge from experts or from experience?
- How can the knowledge be debugged, maintained, and improved?

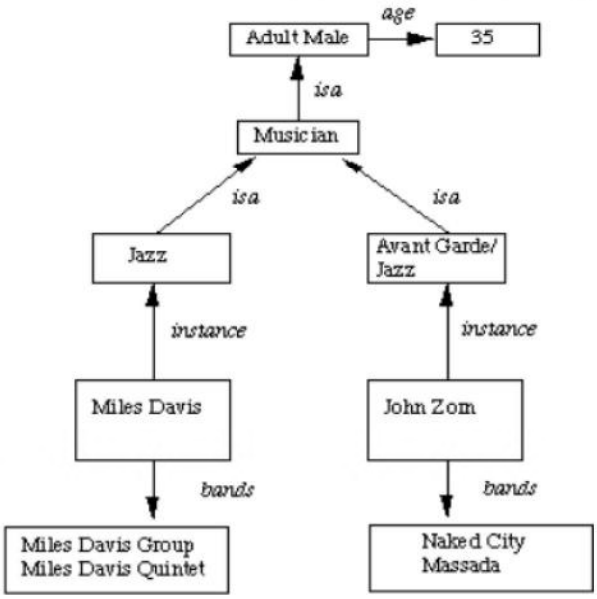


Fig: Inheritable Knowledge representation

Refer to the above diagram…

- **Boxed nodes** - objects and values of attributes of objects.
- **Values**- can be objects with attributes and so on.
- **Arrows**- point from object to its value.

- This structure is known as a slot and filler structure, semantic network or a collection of frames.

Inheritable knowledge

Relational knowledge is made up of objects consisting of

- attributes
- corresponding associated values.

We extend the base more by allowing inference mechanisms:

Property inheritance

- " Elements inherit values from being members of a class. "
- data must be organised into a hierarchy of classes (shown in above figure).

In summary, the main issue in knowledge representation is listed below…

1.Selection of Important Attributes

- How to find important attribute.
- Are there any attributes that occur in many different types of problem?
- There are two instance and **isa** and **each** is important because each supports property inheritance.

2.Relationships

- What about the relationship between the attributes of an object, such as, inverses, existence, techniques for reasoning about values and single valued attributes? We can consider an example of an inverse in

band(John Zorn,Naked City)

This can be treated as John Zorn plays in the band *Naked City* or John Zorn's band is *Naked City*.

3.Granularity

- At what level should the knowledge be represented and what are the primitives. Choosing the Granularity of Representation Primitives are fundamental concepts such as holding, seeing, playing and as English is a very rich language with over half a million words it is clear we will find difficulty in deciding upon which words to choose as our primitives in a series of situations.

Hence,

1. What are the primitives and at what level should the knowledge be represented?
2. What should be the number (small or large) of low-level primitives or high-level facts?

High-level facts may be insufficient to draw the conclusion while Low-level primitives may require a lot of storage.

For example: consider the following facts:
"John spotted Alex".

Now, this could be represented as "Spotted (agent(John), object (Alex))"
Such a representation can make it easy to answer questions such as: Who spotted Alex?

Suppose we want to know : "Did John see Sue?"
Given only one fact, user cannot discover that answer.

Hence, the user can add other facts, such as "Spotted (x, y) → saw (x, y)"

4. Representing sets of objects.

There are some properties of objects which satisfy the condition of a set together but not as individual;

Example: Consider the assertion made in the sentences:
"There are more sheep than people in Australia", and "English speakers can be found all over the world."
These facts can be described by including an assertion to the sets representing people, sheep, and English.

5. Finding the right structure as needed

To describe a particular situation, it is always important to find the access of right structure. This can be done by selecting an initial structure and then revising the choice.

Knowledge Representation Systems

Knowledge representation system is a formalized structure and sets of operations that embodies the descriptions, relationships, and procedures provided in an artificial Intelligence system. The process considers how to represent the knowledge in the system such that it can be used to solve problems.

Components of knowledge Representation system

The knowledge representation function contains the following components.

- Perception
- Learning
- KR and Reasoning
- Planning and Execution

Perception helps in extracting the information and can be helpful in telling us the status of AI system. It can **detect** any irregularity in the system and make us ready to decide whether an AI system has the potentiality of damage or not.

Learning component captures the data which are already sensed by the perception component. Learning component tries to enable the computer to learn just like human instead of always programming it. This component solely tries to focus on how to self-improve the AI system.

KR and reasoning are used in AI to acquire knowledge in the smartest way. It focuses on the behaviour of an AI agent and make sure that it more or less behaves like human. It is used to formalize the knowledge in the knowledge base.

Planning and execution try to find the optimal solution of the current state and tries to understand the impact of the same. Now it tries to seek out the solution that the final state holds and then it will try to terminate the entire process with a solution here itself.

The below diagram shows how the process of knowledge representation works.

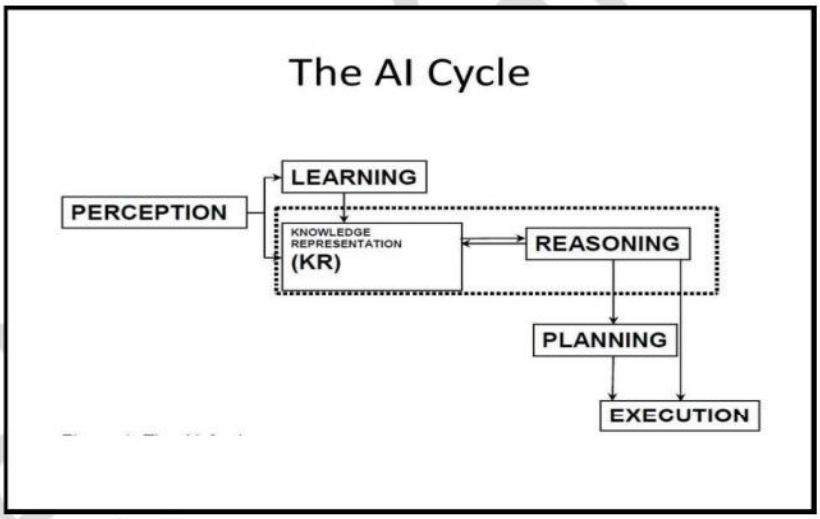


Fig: knowledge representation system

Perception Block

This will help the AI system gain information regarding its surroundings through various sensors, thus making the AI system familiar with its environment and helping it interact with it. These senses can be in the form of typical structured data or other forms such as video, audio, text, time, temperature, or any other sensor-based input.

Learning Block

The knowledge gained will help the AI system to run the deep learning algorithms. These algorithms are written in the learning block, making the AI system transfer the necessary information from the perception block to the learning block for learning (training).

Knowledge and Reasoning Block

We use the knowledge, and based on it, we reason and then take any decision. Thus, these two blocks are responsible for acting like humans go through all the knowledge data and find the relevant ones to be provided to the learning model whenever it is required.

Planning and Execution Block

These two blocks though independent, can work in tandem. These blocks take the information from the knowledge block and the reasoning block and, based on it, execute certain actions. Thus, knowledge representation is extremely useful for AI systems to work intelligently.

Properties of Knowledge Representation System

The knowledge representation system represents the various types of knowledge and it has certain properties that can help us in assessing the system which are as follows

Representational Adequacy

A major property of a knowledge representation system is that it is adequate and can make an AI system understand, i.e., represent all the knowledge required by it to deal with a particular field or domain OR it has enough to express the knowledge needed to solve the problems.

Inferential Adequacy

The knowledge representation system is flexible enough to deal with the present knowledge to make way for newly possessed knowledge.

The ability to represent all of the kinds of inferential procedures that manipulate the representational structures in such a way as to derive new structures corresponding to new knowledge inferred from old.

Inferential Efficiency

The representation system cannot accommodate new knowledge in the presence of the old knowledge, but it can add this knowledge efficiently and in a seamless manner.

For instance, by incorporating into the knowledge structure additional information that can be used to focus the attention of the inference mechanisms in the most promising directions

Acquisitional Efficiency

The final property of the knowledge representation system will be its ability to gain new knowledge automatically, helping the AI to add to its current knowledge and consequently become increasingly smarter and productive.

Types of Knowledge Representation Systems:

- **Semantic Nets**
- **Frames**
- **Conceptual Dependencies**
- **Scripts, Rule Based Systems**
- **Propositional Logic**
- **Predicate Logic**

Semantic Nets

Semantic networks are basically graphic depictions of knowledge composed of nodes and links that show hierarchical relationships between objects. The nodes are interconnected by links or arcs. These arcs show the relationships between the various objects and descriptive factors. Some of the most common arcs are of the is-a or has-a type.

Is-a is used to show class relationship; that is, an object belongs to a larger class or category of objects. Has-a links are used to identify characteristics or attributes of the object nodes. Other arcs are used for definitional purposes.

Semantic networks can show inheritance. Semantic nets are a visual representation of relationships, and can be combined with other representations.

A semantic net is mostly a guided/directed or unguided network graph. It consists

- **Vertices/nodes:** Represent concepts. In these network graphs, nodes appear in form of circles or ellipses or even rectangles that represent objects (Physical objects, Concepts, or Situations).
- **Edges/links:** Represent semantic relations between those concepts. Links appear as lines/arrows to express the relationships between objects. The arrowhead is being used in guided/directed graphs.
- **Link labels:** Specify relations between those nodes.

It is being used to

- Represent data
- Reveal structures (relations between the objects, proximity, and relative importance)
- Support conceptual edition
- Support navigation

Components of Semantic Networks:

- **Lexical Components** - Consists of:
  - **Nodes:** represent the object or concept.

- **Links:** Denoting relation between nodes.
- **Labels:** Denoting particular objects & relations.

- **Structural Component** - Here the links and nodes form a directed graph wherein the labels are placed on the link and nodes.
- **Semantic Component** - The meanings here are related to the links and labels of nodes, whereas the facts are dependent on the approved areas.
- **Procedural Part** - The creation of new links and nodes is permitted by constructors, whereas the destructors are responsible for the removal of links and nodes.

Types of semantic networks

There are six most widely used types of semantic networks:

1) **Definitional network**

Definitional network deals with the relations between a newly defined subtype, and a concept type. A producing network is known as a generalization hierarchy. It supports the inheritance rule for duplicating attributes.

2) **Assertional network**

Assertional network is intended to state recommendations. The data in an assertion network is thought to be unexpectedly genuine, unless it is unequivocally marked with a modal administrator.

Unlike definitional network, the information in an assertional network is assumed to be contingently true, unless it is explicitly marked with a modal operator. Some assertional networks have been proposed as models of the conceptual structures underlying natural language semantics.

3) **Implicational network**

Implicational network is used as the primary relation for connecting nodes. They may be used to explain patterns of beliefs, causality, or deductions.

Implicational networks emphasize implication, they are capable of expressing all the Boolean connectives by allowing a conjunction of inputs to a propositional node and a disjunction of outputs.

4) **Executable network**

Executable network incorporates some techniques, for example, such as attached procedures or marker passing which can perform path messages, or associations, and searches for patterns.

5) **Learning network**

Learning network constructs or expands its representation by securing information. For example, the new information may change the old system by including and excluding nodes and arcs, or by changing numerical qualities called weights, and connected with the arcs and nodes.

Or,

Build or extend their representations by acquiring knowledge from examples. The new knowledge may change the old network by adding and deleting nodes and arcs or by modifying numerical values, called weights, associated with the nodes and arcs.

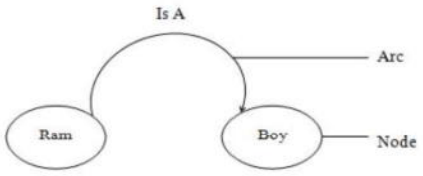
6) Hybrid network

Combine two or more of the previous techniques, either in a single network or in separate, but closely interacting networks.

Hybrid network has been clearly created to implement ideas regarding human cognitive mechanisms, while some are actually created generally for computer performance.

Example: Create a Semantic Net

Example-1: Ram is a boy.



Figure

Example-2: A simple example of a semantic net is shown in Figure below.

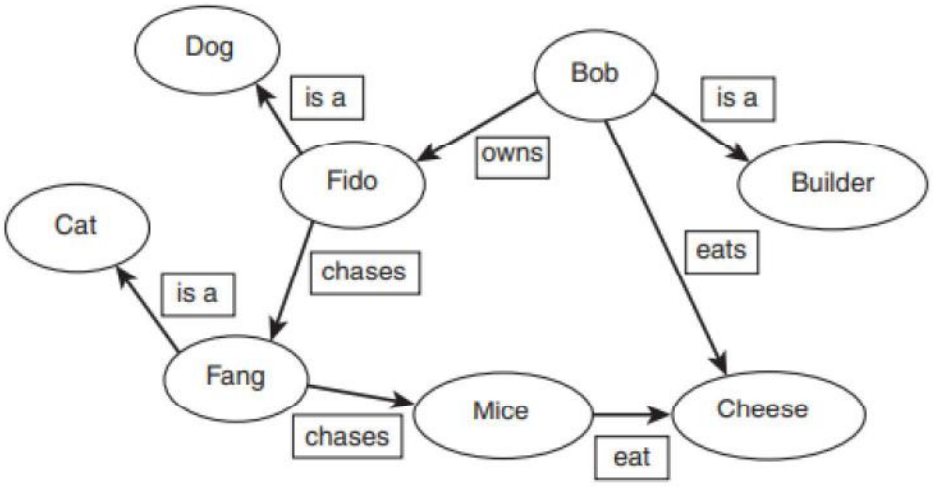


Figure 3.1 A simple semantic net

Note that in this semantic net, the links are arrows, meaning that they have a direction. In this way, we can say from the diagram that Fido chases Fang, not that Fang chases Fido. It may be that Fang does chase Fido as well, but this information is not presented in this diagram.

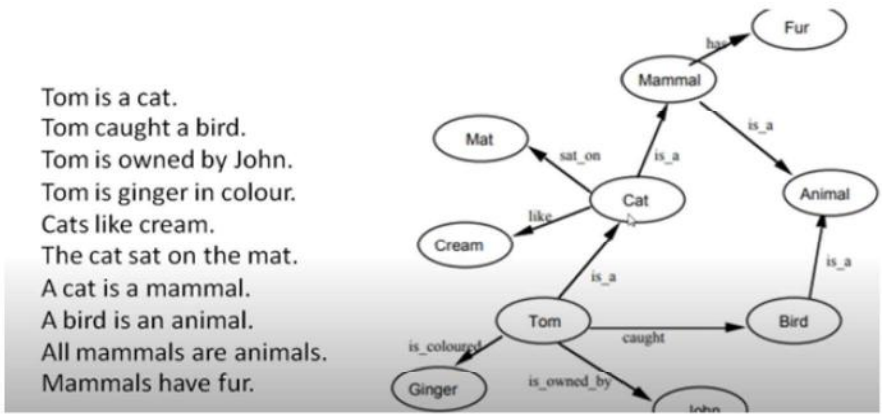
Note that in our semantic net we have represented some specific individuals, such as Fang, Bob, and Fido, and have also represented some general classes of things, such as cats and dogs. The specific objects are generally referred to as instances of a particular class. Fido is an instance of the class dog. Bob is an instance of the class Builder.

It is a little unclear from Figure above whether cheese is a class or an instance of a class. This information would need to be derived by the system that is manipulating the semantic net in some way. For example, the system might have a rule that says "any object that does not have an 'is-a' relationship to a class is considered to represent a class of objects." Rules such as this must be applied with caution and must be remembered when building a semantic net.

An important feature of semantic nets is that they convey meaning. That is to say, the relationship between nodes and edges in the net conveys information about some real-world situation. A good example of a semantic net is a family tree diagram. Usually, nodes in these diagrams represent people, and there are edges that represent parental relationships, as well as relationships by marriage.

Each node in a semantic net has a label that identifies what the node represents. Edges are also labelled. Edges represent connections or relationships between nodes. In the case of searching a dictionary for a page that contains a particular word, each node might represent a single page, and each edge would represent a way of getting from one page to another.

Example-3:

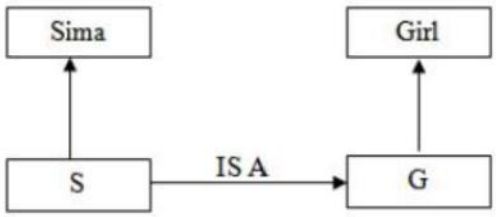


Semantic network by using Instances

In this case we can create one instance of each object. In instance based semantic net representations some keywords are used like: IS A, INSTANCE, AGENT, HAS-PARTS etc.

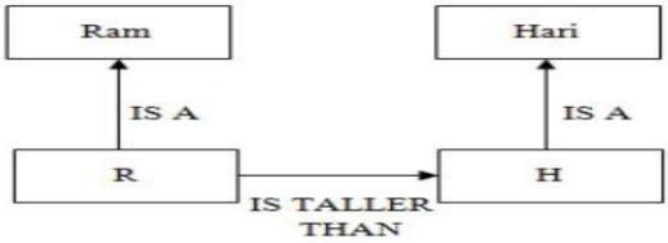
Consider the following examples:

1. Suppose we have to represent the sentence "Sima is a girl".

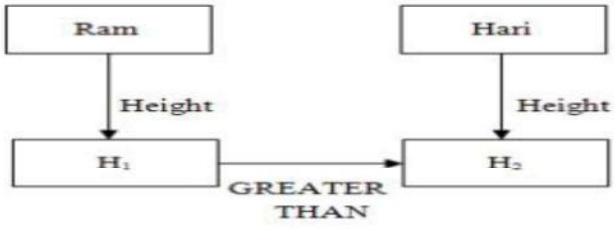


Figure

2. Ram is taller than Hari



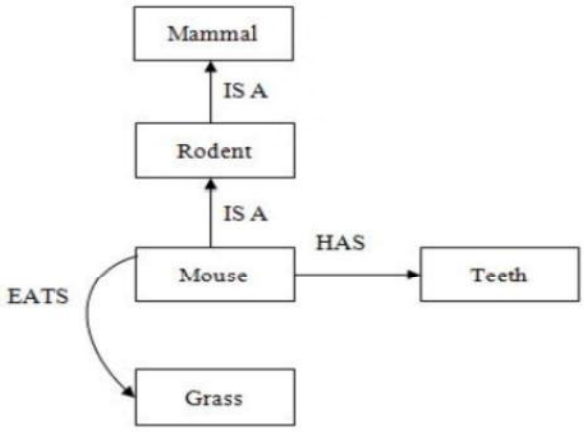
It can also be represented as



(b)

3. "Mouse is a Rodent and Rodent is a mammal. Mouse ha s teeth and etas grass". Check whether the sentence mammal has teeth is valid or not.

(c)



Frame based knowledge representation

What is a frame?

- A frame is a data structure with typical knowledge about a particular object or concept.
- It is a development of semantic nets and allow us to express the idea of inheritance.
- Frames, first proposed by Marvin Minsky in the 1970s (Minsky, 1975), are used to capture and represent knowledge in a frame-based expert system.
- A Frame System consists of a set of frames (or nodes), which are connected together by relations. Each frame describes either an instance or a class.
- Each frame has one or more slots, which are assigned slot values. Rather than simply having links between frames, each relationship is expressed by a value being placed in a slot.
- These slots may be of any type and sizes. Slots have names and values which are called facets.
- **Facets:** Facets are features of frames which enable us to put constraints on the frames. Example: IF-NEEDED facts are called when data of any particular slot is needed.
- A frame may consist of any number of slots, and a slot may include any number of facets and facets may have any number of values.
- A frame is also known as slot-filter knowledge representation in artificial intelligence.

Example 1: Boarding passes shown in Figure below represent typical frames with knowledge about airline passengers.

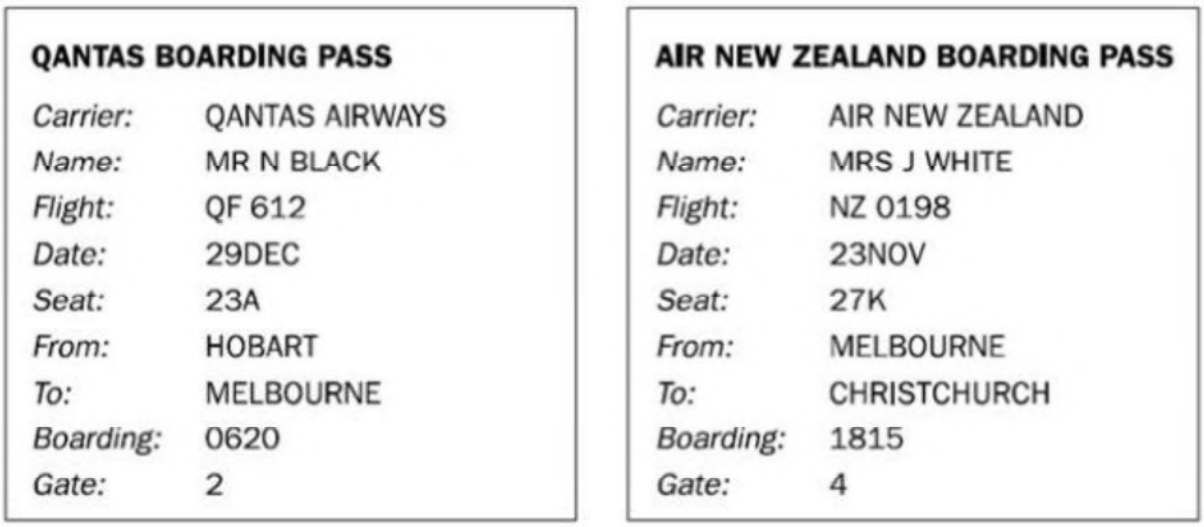


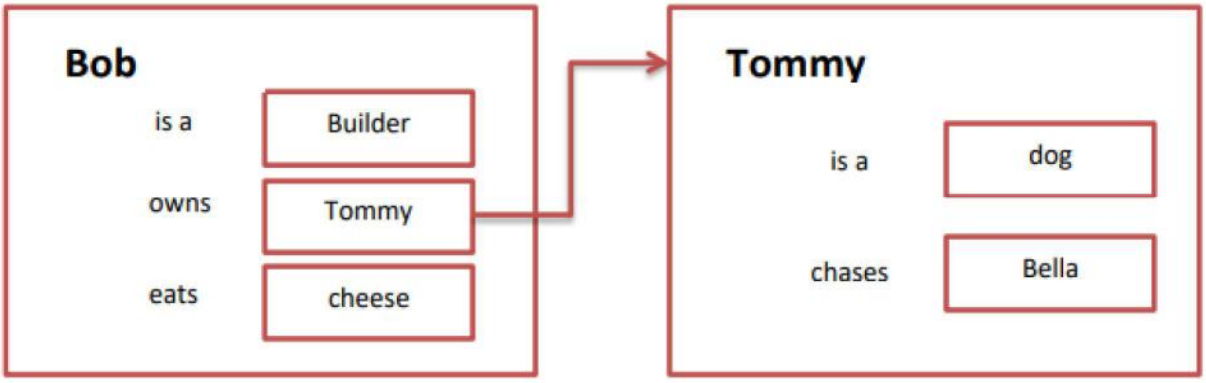
Figure: Boarding pass frame

- Both frames have the same structure.
- Each frame has its own name and a set of attributes, or slots, associated with it.
- Carrier, name, flight ,date are slots in the frame Qantas boarding pass.
- Each attribute or slot has a value attached to it.
- In Figure:(a) above, slot **Carrier** has value **QANTAS AIRWAYS** and slot **Gate** has value 2.
- In some cases, instead of a particular value, a slot may have a procedure that determines the value.

Example 2:-

Frame Name	Slot	Slot Value
Bob	is a	Builder
	owns	Tommy
	eats	Cheese
Tommy	is a	dog
	chases	Bella
Bella	is a	Cat
	chases	Mice
Mice	eat	cheese
Cheese		
Builder		
Dog		
Cat		

We can also represent this frame system in a diagrammatic form such a below:



When we say, "Tommy is a dog" we really mean, "Tommy is an instance of the class dog" or "Tommy is a member of the class dogs".

Why are Frames useful?

The main advantage of using frame-based systems for expert systems is that all information about a particular object is stored in one place.

Why is it necessary to use frames?

- Frames provide a natural way for the structured and concise representation of knowledge.
- In a single entity, a frame combines all necessary knowledge about a particular object or concept.
- A frame provides a means of organizing knowledge in slots to describe various attributes and characteristics of the object.
- If we are searching for knowledge about Qantas frequent flyers, then we want to avoid the search through knowledge about Air New Zealand or British Airways passengers.
- In this situation, we need frames to collect the relevant facts within a single structure.
- Basically, frames are an application of object-oriented programming.

Inheritance in Frames:

Example: 1

We might extend our frame system with the following additional information:

- Dogs chase cats
- Cats chase mice

Now, we do not need to state explicitly that Tommy chases Bella or that Tommy chases mice. We can inherit this information because Bella is an instance of the class cats, and Tommy is an instance of class dogs.

Example 2:

- Mammals breathe
- Dogs are mammals
- Cats are mammals

Here we have created superclass mammals, of which dogs and cats are subclasses. Hence, we do not need to explicitly say that cats and dogs breathe because we can inherit this information. Similarly, we do not need to express explicitly that Tommy and Bella breathe as they are instances of the class dogs and cats and therefore, they inherit from those classes (superclasses).

Example 3:

Let's take a fact:

- Mammals have four legs
- Now, this is not true as humans do not have four legs.

Or let's say Tommy has an unfortunate accident and now has only three legs.

This information might be expressed as:

Frame Name	Slot	Slot Value
Mammals	*no. of legs	four
Dog	subclass	mammal
Cat	subclass	mammal
Tommy	is a	dog
	number of legs	three
Bella	is a	cat

We have used an asterisk (*) to indicate that the value for the "number of legs" slot for the mammal class is a default value and can be overridden.

Multiple Inheritance in Frame System:

It is possible for a frame to inherit properties from more than one frame, i.e., a class can be a subclass of two super-classes and an object can be an instance of more than one class. This is known as Multiple Inheritance.

Example:

Frame Name	Slot	Slot Value
Human	subclass	Mammal
	No. of legs	two
Builder	builds	houses
Bob	is a	Human
	is a	Builder

Hence, we can inherit the following information about Bob:

- He has two legs
- He builds houses

In some cases, we encounter conflicts, where multiple inheritance leads us to conclude contradictory information about a frame.

Example:

Frame Name	Slot	Slot Value
Cheese	is	smelly
Things wrapped in foil	is	not smelly
Cheddar	is a	Cheese
	is a	Things wrapped in foil

Here, cheddar is a type of cheese and that it comes wrapped in foil. Cheddar should inherit its smelliness from the cheese class, but it also inherits non-smelliness from the Things wrapped in foil class.

Thus, we need a mechanism to decide which features to inherit from which superclass.

- One simple method is to say that conflicts are resolved by the order in which they appear. So, if a fact is established by instances and then that fact is contradicted by inheritance, the first fact is kept because it appeared first and the contradiction is discarded.
- However, it would be better to build the frame system such that conflicts of this kind cannot occur.

Conceptual Dependencies

- Conceptual dependency (CD) is a theory of natural language processing which mainly deals with representation of semantics of a language.
- It was developed by Roger C Schank in 1975 as a part of a natural language comprehension project.
- It is a theory of how to represent the meaning of natural language sentences in which it helps to draw inferences from the sentences and it is independent of the language in which the sentences were originally stated.
- CD representation of a sentence is not built using words in the sentence rather built using conceptual primitives which give the intended meanings of words.
- Since it represents meaning of sentence using primitives but not exact words in a sentence:
  - It helps in drawing inferences
  - It is independent of the language

Conceptual dependency theory was based on two assumptions:

1. If two sentences have the same meaning, they should be represented the same, regardless of the particular words used i.e. It states that different sentence which has the same meaning should have some unique representation.
2. Information implicitly stated in the sentence should be represented explicitly.

Main Goals of Conceptual Dependency:

1. It captures the implicit concept of a sentence and makes it explicit.
2. It helps in drawing inferences from sentences.
3. For any two or more sentences that are identical in meaning. It should be only one representation of meaning.
4. It provides a means of representation which are language independent.
5. It develops language conversion packages.

The main ideas of this system can be summarized as follows:

I. Two phrases or sentences having equivalent meanings must have the same internal representation.
II. Every action should be expressed in terms of certain primitives. For example, a primitive for 'drink' could be 'ingest', which could also be used for 'swallow' and 'eat'.
III. All the information implied in a phrase or sentence must be made explicit in the internal representation.

The meaning of a phrase is represented by a schema. This consists of four kinds of nodes or categories:

PP-- Real world objects.{Picture Producers}
ACT-- Real world actions. {One of the CD primitives}
PA-- Attributes of objects.{Picture Aiders}
AA-- Attributes of actions.{Action aiders}
T-- Times.
LOC-- Locations.

a) PP (Picture Production) equivalent to **nouns** (Only physical objects are physical producers).
b) ACT (Action) equivalent to **verbs** (Actions are done by an actor to an object similarly a person who performs some action is known as actor)
c) PA (Picture Aider) equivalent to **adjectives** (Modifiers of PP) (PA practically serve PP by defining the characteristics.)
d) AA (Action Aider) equivalent to **adverbs** (Modifiers of ACT) (These serve as modifiers of actions; the actor PROPEL has a speed factor associated with it which is an action aider.)

Schank has developed the use of a set of four categories of concepts in a sentence. These are Objects, Actions, Modifiers of Actions, and Modifiers of Objects.

Dependencies or relations are represented by:

1. **<==>:** This means mutual dependence between two concepts
2. **---->:** This means one way dependence between an ACT and a PP or between a PP and a PA
3. **<==:** This means one way dependence between two PPs.

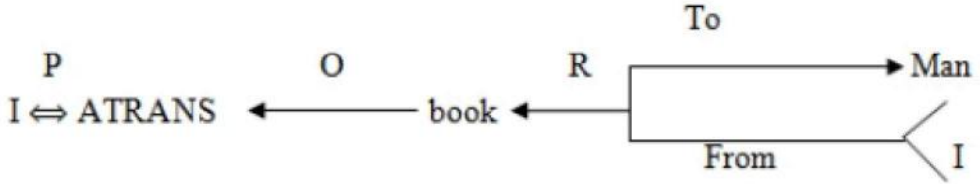
Some of the major primitive actions are:

Primitive Action **Explanation**

ATRANS: - Transfer of an abstract relationship (Eg: give,took)(transfer of any object)
PTRANS: - Transfer of the physical location of an object(Eg: go)
PROPEL: - Application of physical force to an object (Eg: push,pull)(apply physical force)
MOVE: - Movement of a body part by its owner (eg : kick,push)
GRASP: - Grasping of an object by an actor(Eg: throw)(perform action in any object)
INGEST: - Ingesting of an object by an animal (Eg: eat) (take anything or take out)
EXPEL: - Expulsion of something from the body of an animal (cry,sweat) (take out anything from body)
MTRANS: - Transfer of mental information(Eg: tell,see) (mental transfer i.e. communicating)
MBUILD: - Building new information out of old(Eg: decide) (think new thing)
SPEAK: - Production of sounds(Eg: say)
ATTEND: - Focusing of sense organ toward a stimulus (Eg: listen)

Example: I gave book to the man

⇨ The CD representation of this is as follows



Where;

- **←:** Is the direction of dependency
- **⇔(Double arrow):** It indicates two-way link between actor and action.
- **P:** Past Tense
- **ATRANS:** One of the primitive acts used by the theory
- **O:** The objective case relation
- **R:** Recipient case Relation

The main goal of CD representation is to capture the implicit concept of a sentence and make it explicit. In normal representation of the concepts, besides actor and object, other concepts of time, location, source and destination are also mentioned. Following conceptual tenses are used in CD representation.

1) O : Object case relationship
2) R : Recipient case relationship
3) P : Past
4) F : Future
5) Nil : Present
6) T : Transition
7) Ts : Start Transition
8) Tf : Finisher Transition
9) K : Continuing
10) ? : Interrogative
11) / : Negative
12) C : Conditional

Rule 1: PP ⇔ ACT

It describes the relationship between an actor and the event he or she causes.

- This is the two-way dependency, since neither actor nor event can be considered primary.
- The latter P in the dependency link indicate past tense.

Example: ram ran

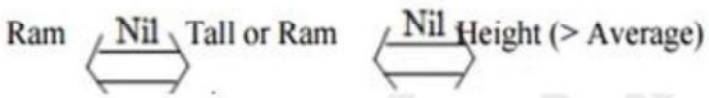
p
Ram⇔PTRANS
Where p is paste tense

Rule 2: PP⇔ PA

It describes the relationship between a PP and PA
where PA indicates one state/characteristics of PP such as height, health etc

Example 1: John is fat
CD representation: John ⇔ weight (>80)
Example 2:

Ram is tall



Rule 3: PP ⟺ PP

It describes the relationship between two PPs where one PP is defined by other.

E.g. Ram is a doctor

Ram ⟺ (Nil) Doctor

Rule 4: PP ↑ PA or PA ↓ PP

It describes the relationship between the PP and PA, where PA indicates one attributes of PP.

E.g. A nice boy is a doctor

Boy ⟺ (Nil) Doctor

↑

Nice

Rule 5: PP

⇑

PP

It describes the relationship between **2** PP's where one PP is the owner of another PP.

E.g. Ram's Cat

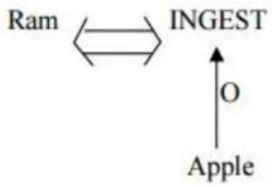
Cat

⇑

Ram

Rule 6: Act ← O PP Where O: Object

It describes the relationship between the PP and ACT. Where PP indicates the object of that action. E.g. Ram is eating an apple.

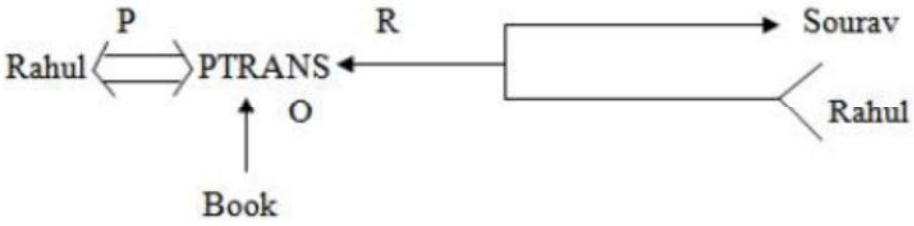


Rule 7: ACT ← R → Recipient; Donner

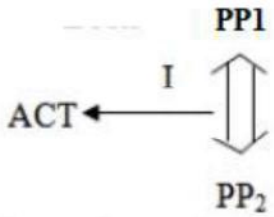
(R: Recipient)

Here one PP describes the recipient and another PP describes the donner

E.g. Rahul gave a book to sourav.



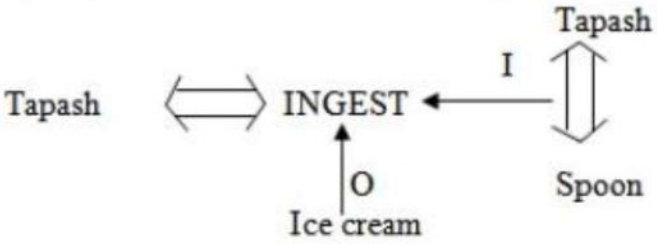
Rule 8:



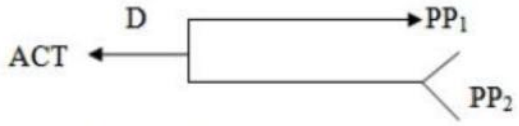
(I: Instrument used in the action)

Here PP_1 indicates the agent and PP_2 indicates the object that is used in the action.

E.g. Tapash ate the ice cream with the spoon.

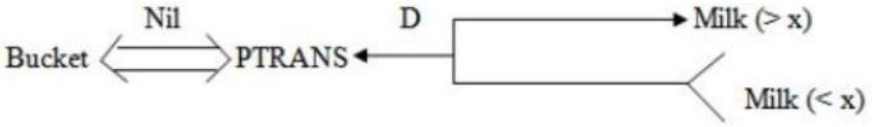


Rule 9:



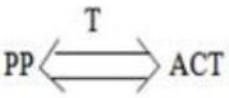
Here D indicates destination, PP_1 indicates destination and PP_2 indicates the source.

E.g. the bucket is filled with milk.



xindicates the average milk and the source i.e. bucket is dry which is hidden.

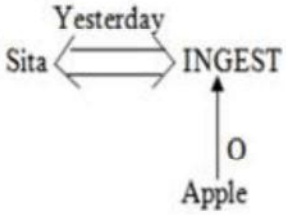
Rule 10:



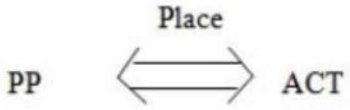
(T: Time)

It describes the relationship between a conceptualization and the time at which the event is described occurs.

E.g. Sita ate the apple yesterday.

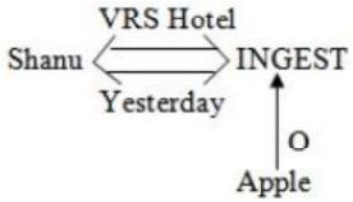


Rule 11:

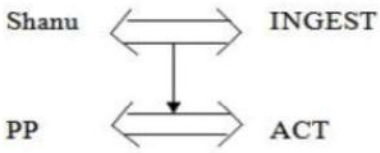


It describes the relationship between a conceptualization and the place at which it is occurred.

E.g. Shanu ate the apple at VRS hotel yesterday

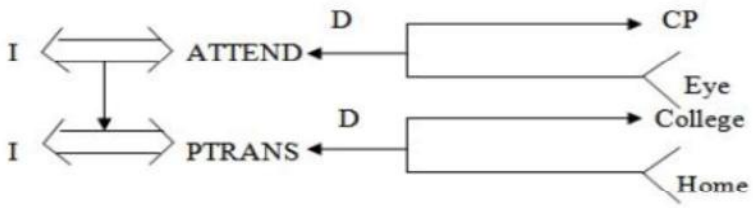


Rule 12:



It describes the relationship between one conceptualization with another.

E.g. while I was going to college, I saw a snake

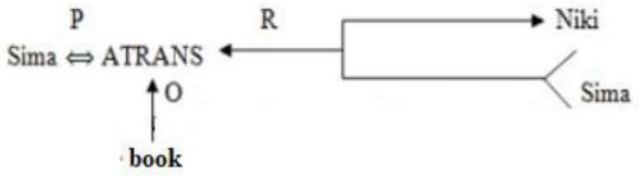


(Where CP: Conscious Processor i.e. the combination of all sense organs like eye, ear, nose etc.)

By using the above rules we can represent any sentence. Let us visualize few examples on conceptual dependency.

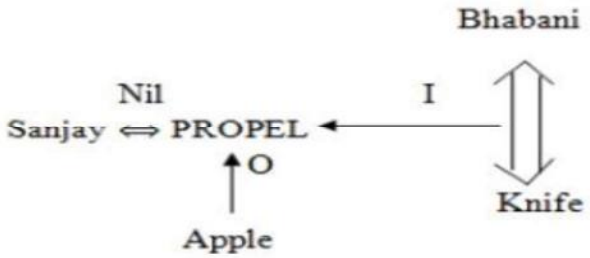
Other Examples:

1) Sima gave a book to Niki

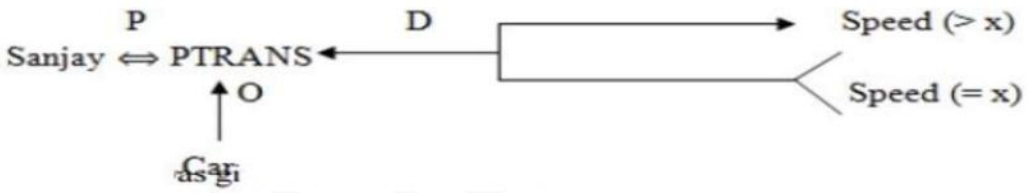


Where O: Object, P: Past Tense, R: Recipient, Sima: PP, Book: PP, Niki: PP, ATRANS: give

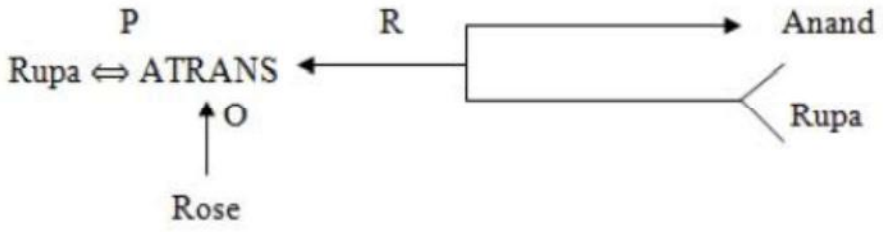
2) Bhabani cuts an apple with a knife



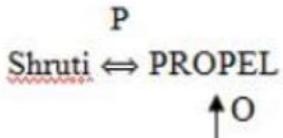
3) Sanjay drove the car fast



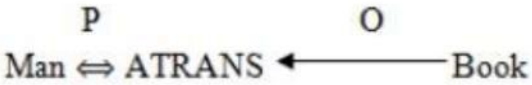
4) The rose was given by Rupa to Anand



5) Shruti pushed the door.



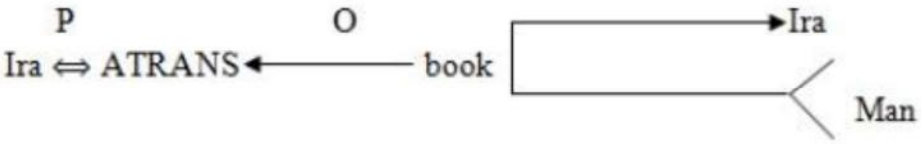
6) The man took a book



Here man is the doctor and book is the object of the action took.

7) My grandfather told me a story

8) Ira gave the man a dictionary



Problems with CD Representation

1. It is difficult to
   - construct original sentence from its corresponding CD representation.
   - CD representation can be used as a general model for knowledge representation, because this theory is based on representation of events as well as all the information related to events.

2. Rules are to be carefully designed for each primitive action in order to obtain semantically correct interpretation.

3. Many verbs may fall under different primitive ACTs, and it becomes difficult to find correct primitive in the given context.

4. The CD representation becomes complex requiring lot of storage for many simple actions

5. For example, the sentence "John bet Mike that Indian cricket team will win incoming world cup" will require huge CD structure

Script

Script is one of the types of knowledge representation system in artificial intelligence. A script is a structured representation; describing a stereotyped sequence of events in a particular context.

Scripts are used in natural language understanding systems to organize a knowledge base in terms of the situations that the system should understand. Scripts use a frame-like structure to represent the commonly occurring experience like going to the movies eating in a restaurant, shopping in a supermarket etc. That is scripts are designed to describe the stereotyped events like going to a restaurant or going to watch a movie and so on.

For example, when I say "I went to Indroda Park (a typical natural zoo) with kids", you may assume the complete sequence of buying tickets, parking car in the slot, taking picnic related items from the car, move around, watch and appreciate animals, take rest during the lunch time (or breakfast time if it is evening) and enjoy food that we brought with us, return back to car and then move out of zoo.
Hence, Script is a mechanism for representing knowledge about sequence of such events and inferring from the same.

A typical script structure is shown in the figure (a): - below which is a very simplified representation for watching a match.

Table (a): - A script for watching a cricket match

Script: watching a match	Various Scenes
Track: Cricket match **Props:** • Tickets • Seat • Match **Roles:** • Person (who wants to Watch a match) – P • Booking Clerk – BC • Security personal – SP • Ticket Checker - TC **Entry Conditions:** • P wants to watch match • P has a money **Results:** • P saw a match • P has less money • P is happy (if his team has won) or not (if his team has lost) or some other problem at stadium	***Scene 1: Going to a stadium*** • P PTRANS P to the stadium • P ATTEND eyes to ticket counter ***Scene 2: Buying ticket*** • P PTRANS P to ticket counter • P MTRANS ticket requirement to BC • P MTRANS stand information to BC • BC ATRANS ticket to P ***Scene 3: Going inside stadium and sitting on a seat*** • P PTRANS P into Stadium • S PMOVE security check P • TC ATTEND eyes on ticket POSSby P • TC MOVE to tear part of ticket • TC MTRANS (showed seat) to P • P PTRANS P to seat • P MOVES P to sitting position ***Scene 4: Watching a match*** • P ATTEND eyes on match • P MBUILD (moments) from the match ***Scene5: Exiting*** • P PTRANS P out of Stadium

In this example, Names and other items are filled as and when needed. For example, when we say that Jay went to watch a match, Jay will replace by P. If you try to recall our description of frames, you can see that there is some similarity. There are places which resembles slots.

The table (a) above; script has five scenes, two entry conditions, two props, four roles and a typical track. Each script contains those components (depicted in the table (b) below).

- Entry conditions are must for entering the script.
- Results are the outcomes of the script.
- Props and roles are items and people involved in the process while scenes are there to describe sequence of events.

Table (b) Components of a script

Entry Conditions:	Conditions that must be satisfied for execution of the script. Whenever a script is referred, one can assume the preconditions to be true. That is, it must be true before the events described in the script can occur For example, Jay went to watch a cricket match, it is also concluded that he has money to buy tickets and he is also a cricket fan. Even when not explicitly mentioned, entry conditions can safely assume to be true. Similarly, another example in a restaurant script the entry condition must be the customer should be hungry and the customer has money.
Results:	The Conditions that will be true after exit. This is a general (and thus default) assumption. It might be false under exceptional circumstances. For example, when there is rain and the match is abandoned, watching does not happen. Happy may also be false.
Props:	Objects involved in the script i.e. It describes the inactive or dead participants in the script e.g. for watching cricket match Tickets, seats etc and similarly for example , Shopping in supermarket script, the probes may be clothes, sticks, doors, tables, bills etc.
Roles:	Persons involved in the script or the actions that the individual participants perform. Hence, in conclusion; it is **expected actions** of the major participants in the events described in the script. E.g., In a restaurant script the scenes may be entering, ordering etc.
Track:	These represent variations on **general theme** or pattern represented by a particular script. Specialization of the script derived from a general pattern. A general pattern may inherit multiple tracks. That means multiple such tracks look quite similar but they have their own individually different scenes or other items associated. For example, watching a football match might contain referees, linemen, and so on while a detailed cricket match might have a wicket keeper, umpires, a third umpire and so on. So, in conclusion, it specifies particular position of the script e.g. In a supermarket script the tracks may be cloth gallery, cosmetics gallery etc.
Scenes:	The sequence of events following a general default path. Events are represented in CD form but mentioned as a semi-CD form, just describing the ACT and rest in English for pedagogy purpose.

There is default event related information provided. For example, it is assumed that the entry in the stadium is done after tickets are obtained from outside ticket counter. That is a default information. In case of no other information is available, they are assumed to be true.

It is not always the case that we pick up default values for scenes. It is also possible that the tickets are purchased online and that step is already eliminated (so it is to be removed from the scrip inserted in the place where the statement is introduced).

It is also possible that due to tight security some belongings are to be invited to be kept in the locker room which we have not discussed here but possible (so to be added to whatever default is provided by the Script).

The Point is, what we have described is a default assumption which might change. In fact, it is possible to assume multiple paths for a given event and write all of them in a script. For example one might not sit till the match ends. If he finds his team is surely going to lose, he might leave early.

Similarly in a restaurant script the last scene is about paying the bills. If the customer dislikes the food or dislikes the service, he follows the non-payment path. For example, if we encounter statement, "Jay asked for Italian Pizza with double cheese. He waited for 15 minutes, got angry and went away". You can easily understand that he has followed the alternate path and thus neither eating nor payment part is considered. That means it is possible to have default values which might change if we have evidence to the contrary. Thus when information is provided, other than default values are taken. However, the script is quite different from frames. Different objects representing same class by frame only differ in the value of their attributes but the attributes are all the same.

Example-2 : A movie Ticket Script.

<table>
<tr><th>Script: watch movie</th><th>Various Scenes</th></tr>
<tr><td rowspan="4">Track: CINEMA HALL
Props:
Ticket, snacks, chair, money, Ticket, chart

Roles:
• Customer(c),
• Ticket seller(TS),
• Ticket Checker(TC),
• Snacks Sellers (SS)

Entry condition:
• The customer has money
• The customer has interest to watch movie.

Results:
• The customer is happy
• The customer has less money</td><td>Scene 1: Entering into the cinema hall
• C PTRANS C into the cinema hall
• C ATTEND eyes towards the ticket counter
• C PTRANS C towards the ticket counters
• C ATTEND eyes to the ticket chart
• C MBUILD to take which class ticket
• C MTRANS TS for ticket
• C ATRANS money to TS
• TS ATRANS ticket to C</td></tr>
<tr><td>Scene 2: Entering into the main ticket check gate
• C PTRANS C into the queue of the gate
• C ATRANS ticket to TC
• TC ATTEND eyes onto the ticket
• TC MBUILD to give permission to C for entering into the hall
• TC ATRANS ticket to C
• C PTRANS C into the picture hall.</td></tr>
<tr><td>Scene 3: Entering into the picture hall
• CATTEND eyes into the chair
• TC SPEAK where to sit
• C PTRANS C towards the sitting position
• C ATTEND eyes onto the screen</td></tr>
<tr><td>Scene 4: Ordering snacks
• C MTRANS SS for snacks
• SS ATRANS snacks to C
• C ATRANS money to SS
• C INGEST snacks e.</td></tr>
<tr><td>Scene5: Exit
• C ATTEND eyes onto the screen till the end of picture
• C MBUILD when to go out of the hall
• C PTRANS C out of the hall</td></tr>
</table>

Example 3: Write a script of visiting a doctor in a hospital

<table>
<tr><th>Script: Visiting a doctor</th><th>Various Scenes</th></tr>
<tr><td rowspan="5">Track: ENT specialist
Props:
Money, Prescription, Medicine, sitting chair, Doctor's table, Thermometer, Stethoscope, writing pad, pen, torch, stature.
Roles:
Attendant (A), Nurse(N), Chemist (C), Gatekeeper(G), Counter clerk (CC), Receptionist(R), Patient(P), Ent specialist Doctor (D), Medicine Seller (M).

Entry condition:
• The patient need consultation.
• Doctor's visiting time on.
Results:
• The patient has less money
• Patient has prescription and medicine.</td><td>Scene 1: Entering into the hospital
• P PTRANS P into hospital
• P ATTEND eyes towards ENT department
• P PTRANS P into ENT department
• P PTRANS P towards the sitting chair</td></tr>
<tr><td>Scene 2: Entering into the Doctor's Room
• P PTRANS P into doctor's room
• P MTRANS P about the diseases
• P SPEAK D about the disease
• D MTRANS P for blood test, urine test
• D ATRANS prescription to P
• P PTRANS prescription to P.
• P PTRANS P for blood and urine test.</td></tr>
<tr><td>Scene 3: Entering into the Test Lab
• P PTRANS P into the test room
• P ATRANS blood sample at collection room
• P ATRANS urine sample at collection room
• P ATRANS the examination reports</td></tr>
<tr><td>Scene 4: Entering to the Doctor's room with Test reports
• P ATRANS the report to D
• D TTEND eyes into the report
• D MBUILD to give the medicines
• D SPEAK details about the medicine to P
• P TRANS doctor's fee
• P PTRANS from doctor's room</td></tr>
<tr><td>Scene5: Entering towards medicine shop
• P PTRANS P towards medicine counter
• P ATRANS Prescription to M
• M ATTEND eyes into the prescription
• M MBUILD which medicine to give
• M ATRANS medicines to P
• P ATRANS money to M
• P PTRANS P from the medicine shop</td></tr>
</table>

Rule Based Systems

In Rule-based systems, knowledge is encoded in the form of facts, goals, and rules and is used to evaluate and manipulate data or it can be said that it is a system that applies human-made rules to store, sort and manipulate data. Such that it mimics human intelligence.

Rule-based system also called production system is a knowledge-based system (KBS) in which the knowledge is stored as a rule; and also called an expert system in RBSs in which the rules come from human experts in a particular domain.

A rule-based system uses rules as the knowledge representation. These rules are coded into the system in the form of if-then statements. This rule consists of two parts if part (also called antecedent (premise or condition) and then part also called consequent (conclusion or action)

Syntax:

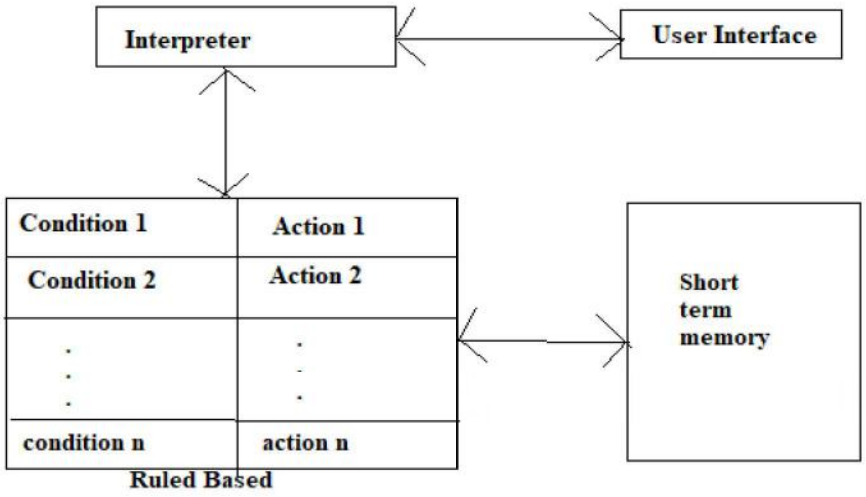
If <Premise> Then <action>
Or
IF P THEN Q
which is also equivalent to:
P⇒Q.

Example:

1. If it is raining then open the umbrella.

Hence, the main idea of a rule-based system is to capture the knowledge of a human expert in a specialized domain and embody it within a computer system. Hence, knowledge is encoded as rules.

Components of rule-based system



- **Working memory**: - a small allocation of memory into which only appropriate rules are copied.
- **Rule base**: - the rules themselves, possibly stored specially to facilitate efficient access to the antecedent.
- **Interpreter**: - The processing engine which carries out reasoning on the rules and derives an answer.
- **A user interface or other connection**: - to the outside world through which input and output signals are received and sent.

Propositional Logic, Predicate Logic

Note: - This two knowledge representation system is explained in further detail

Propositional Logic

Logic

Logic can be defined as the proof or validation behind any reason provided. It is simply the 'dialectics behind reasoning'. It is important to include logic in Artificial Intelligence because we want our agent (system) to think and act humanly, and for doing so, it should be capable of taking any decision based on the current situation.

In general, **Logics** are formal languages for representing information such that conclusions can be drawn. E.g., the language of arithmetic

$x + 2 \geq y$ is true iff the number $x + 2$ is no less than the number y
$x + 2 \geq y$ is true in a world where $x = 7, y = 1$
$x + 2 \geq y$ is false in a world where $x = 0, y = 6$

If we talk about normal human behaviour, then a decision is made by choosing an option from the various available options. There are reasons behind selecting or rejecting an option. So, our artificial agent should also work in this manner. Several different forms of logic are used in AI research like propositional logic predicate logic etc.

Propositional logic

Propositional logic is a simple form of logic which is also known as Boolean logic. It is a technique of knowledge representation in logical and mathematical form. A proposition has TRUTH values (0 and 1) which means it can have one of the two values i.e. True or False. It 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.**

There are two types of Propositions:

a. **Atomic Propositions/sentences**
b. **Compound propositions/sentences**

Atomic Proposition:

Atomic propositions are the simple propositions. It consists of a single proposition symbol. We use symbols that start with an uppercase letter and may contain other letters or subscripts, for example: P, Q, R, W1,3 and North. There are two proposition symbols with fixed meanings: True is the always-true proposition and False is the always-false proposition.

Compound proposition:

Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives. Logical connectives are used to connect two simpler propositions or representing a sentence logically. We can create compound propositions with the help of logical connectives

Example-1:

a) "It is raining today, and street is wet."
b) "Ajay is a doctor, and his clinic is in Kathmandu."

Example-2:

If P is a proposition and Q is a proposition, then you can make several compound propositions from them such as:

P∧Q: P and Q
P∨Q: P or Q or both
P⇒Q: P implies Q, also read If P then Q
P⇔Q: P if and only if Q
¬P: Not P, also read It is not the case that P

You can use several of them together as well. Here's one of the De Morgan's laws:
¬(P∧Q) ⇔ ¬P∨¬Q

Propositional logic is used in artificial intelligence for planning, problem-solving, intelligent control and most importantly for decision-making. It is all about Boolean functions and the statements where there are more than just true and false values, includes the certainty as well as uncertainty, it led to the foundation for machine learning models. It is a useful tool for reasoning, but it has limitation because it cannot see inside prepositions and take advantage of relationships among them.

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.**
- 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:

A sentence in any language contains a combination of words like the verb, noun, pronoun, prepositions, etc., similarly Syntax of PL Language also follows a similar rule, and it consists of

1. **Simple undividable statement**
   It represents true or false (not both) and it is Boolean in nature. Upper Case letters A, B, C, P, Q, R are used to represent these statements

   Syntax: A, B, C, P, Q, R

2. **Logical Connectors or operators**
   These connectors are used to connect two statements they are ^, v, →, ↔, ¬
   Syntax: ^, v, →, ↔, ¬
   These connectors are used to represent AND, OR, Implies, bi-conditional and NOT condition.
3. **Complex conditions**
   Complex conditions are handled by connectors within parenthesis.

Logical Connectives:

Logical connectives are used to connect two simpler propositions or representing a sentence logically. There are mainly five connectives, which are given as follows:

1. **Negation:** A sentence such as ¬ P is called negation of P. A literal can be either Positive literal or negative literal.

2. **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.
   So, we can write it as P∧ Q

3. **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 Engineer,
So, we can write it as P ∨ Q.

4. **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
   The implication symbol is sometimes written as ⊃ or →.

5. **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.
   The Biconditional symbol is sometimes written as =.

Following is the summarized table for Propositional Logic Connectives:

word	symbol	example		terminus technicus
not	$\neg$	not A	$\neg A$	negation
and	$\wedge$	A and B	$A \wedge B$	conjunction
or	$\vee$	A or B	$A \vee B$	disjunction
implies	$\rightarrow$	A implies B	$A \rightarrow B$	implication
if and only if	$\leftrightarrow$	A if and only if B	$A \leftrightarrow B$	biconditional

$$
\begin{aligned}
\textit{Sentence} &\rightarrow \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} &\rightarrow \textit{True} \mid \textit{False} \mid P \mid Q \mid R \mid \ldots \\
\textit{ComplexSentence} &\rightarrow (\textit{Sentence}) \mid [\textit{Sentence}] \\
&\mid \neg \textit{Sentence} \\
&\mid \textit{Sentence} \wedge \textit{Sentence} \\
&\mid \textit{Sentence} \vee \textit{Sentence} \\
&\mid \textit{Sentence} \Rightarrow \textit{Sentence} \\
&\mid \textit{Sentence} \Leftrightarrow \textit{Sentence} \\
\text{OPERATOR PRECEDENCE} &: \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow
\end{aligned}
$$

Figure : A BNF (Backus–Naur Form) grammar of sentences in propositional logic, along with operator precedence, from highest to lowest.

Semantics

Semantics define the "meaning" of sentences; i.e., define truth of a sentence in a world. It defines the rules for determining the truth of a sentence with respect to a particular model. Semantics can be used to understand sentences of the language. In propositional logic, a model simply fixes the truth value—true or false for every proposition symbol.

The semantics for propositional logic must specify how to compute the truth value of any sentence, in a given model. This is done recursively. All sentences are constructed from atomic sentences and the five connectives; therefore, we need to specify how to compute the truth of atomic sentences and how to compute the truth of sentences formed with each of the five connectives.

Example1,

An interpretation consists of a function π that maps atoms to {true, false}.
If π(a)=true, we say atom a is true in the interpretation, or that the interpretation assigns true to a.
If π(a)=false, we say a is false in the interpretation.

The interpretation maps each proposition to a truth value. An atomic proposition (a) is true in the interpretation if π(a)=true; otherwise, it is false in the interpretation. The truth value of (a) compound proposition is built using the truth table.

Example2

For Atomic sentences:

True is true in every model and False is false in every model.

For complex sentences: -

we have five rules, which hold for any sub sentences P and Q in any model m (here "iff" means "if and only if"):

- **¬P is true iff P is false in m.**
- **P ∧ Q is true iff both P and Q are true in m.**
- **P ∨ Q is true iff either P or Q is true in m.**
- **P ⇒ Q is true unless P is true and Q is false in m.**
- **P ⇔ Q is true iff P and Q are both true or both false in m.**

The rules can also be expressed with truth tables in Figure below. From these tables, the truth value of any sentence s can be computed with respect to any model m by a simple recursive evaluation.

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Truth Table

In order to maps the truth values of propositions for all combinations that are possible with several logical connectives the **Truth table** is used. The following Truth table depicts the truth values of various combinations of Boolean conditions for Statements P and Q for all the logical connectives.

P	**Q**	**Negation**		**Conjunction**	**Disjunction**	**Implication**	**Bi-conditional**
		¬ P	**¬ Q**	**P ^ Q**	**P v Q**	**P → Q**	**P⇔Q**
True	True	False	False	True	True	True	True
True	False	False	True	False	True	False	False
False	True	True	False	False	True	True	False
False	False	True	True	False	False	True	True

Note:

- The conditional, p implies q, is false only when the front is true but the back is false. Otherwise, it is true.
- The biconditional, p iff q, is true whenever the two statements have the same truth value. Otherwise, it is false.

Tautology

A compound proposition that is always true for all possible truth values of the propositions is called a **tautology**. Or A tautology is sentence having a combination of atomic sentences such that it always gives a truth value independent of the truth or falsity of its constituent atomic sentences.

- A compound proposition that is always false is called a contradiction.
- A proposition that is neither a tautology nor contradiction is called a contingency.

Example-1: $p \vee \neg p$ is a tautology.

p	¬p	p ∨ ¬p
T	F	T
F	T	T

Example-2: p ∧ ¬p is a contradiction.

p	¬p	p ∧ ¬p
T	F	F
F	T	F

Example-3: (P∧Q) ⟹ Q is a tautology

P	Q	$P \wedge Q$	$(P \wedge Q) \implies Q$
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	T

Since the statement (P∧Q) ⟹ Q is true even if one or both of its atomic sentences are false, it is a tautology

Example-4: Show that $(P \to Q) \vee (Q \to P)$ is a tautology.

The truth table for $(P \to Q) \vee (Q \to P)$

P	Q	$P \to Q$	$Q \to P$	$(P \to Q) \vee (Q \to P)$
T	T	T	T	T
T	F	F	T	T
F	T	T	F	T
F	F	T	T	T

Hence, the formula is a tautology.

Validity

A sentence is valid if it is true in all models,

Example:

A∨¬A, A=>A, (A∧ (A=>B)) =>B

Valid sentences are also known as tautologies. Every valid sentence is logically equivalent to true.

Well-formed Formulas (WFFs)

Any expression that obeys the syntactic rules of propositional logic is called a well-formed formula, or **WFF**. Propositional logic uses a symbolic "language" to represent the logical structure, or form, of a compound proposition. Like any language, this symbolic language has rules of syntax that is grammatical rules for putting symbols together in the right way.

The syntax of propositional logic has only three rules:

- Any capital letter by itself is a WFF.
- Any WFF can be prefixed with "¬". (The result will be a WFF too.)
- Any two WFFs can be put together with "∧", "∨", "->", or "⇔" between them, enclosing the result in parentheses. (This will be a WFF too.)

Here are some examples of well-formed formulas, along with brief explanations how these formulas are formed in accordance with the three rules of syntax:

WFF	**Explanation**
A	by rule 1
¬A	by rule 2, since A is a WFF
¬¬A	by rule 2 again, since ¬A is a WFF
(¬A ∧ B)	by rule 3, joining ~A and B
((¬A ∧ B) -> ¬¬C)	by rule 3, joining (¬A ∧ B) and ¬¬C
¬((¬A ∧ B) -> ¬¬C)	by rule 2, since ((¬A ∧ B) -> ¬¬C) is a WFF

Note:

1. The implication symbol (->) is sometimes written as ⊃
2. The Biconditional symbol (⇔) is sometimes written as ≡ and also
3. The negation symbol (¬) is sometimes written as ~.

Following formulas are not well-formed:

Non-WFF	**Explanation**
A~	the ~ belongs on the left side of the negated proposition
(A)	parentheses are only introduced when joining two WFFs with •, ∨, ⊃, or ≡
(A •)	there's no WFF on the right side of the •
(A • B) ⊃ C)	missing parenthesis on the left side
(A • ⊃ B)	cannot be formed by the rules of syntax

Properties of Propositional Logic

Satisfiable: - An atomic propositional formula is satisfiable if there is an interpretation for which it is true.
Tautology: - A propositional formula is valid or a tautology if it is true for all possible interpretations.
Contradiction: - A propositional formula is contradictory (unsatisfiable) if there is no interpretation for which it is true.
Contingent: - A propositional logic can be contingent which means it can be neither a tautology nor a contradiction.

Properties of Operators:

Commutativity:

- $P \wedge Q = Q \wedge P$, or
- $P \vee Q = Q \vee P$.

Associativity:

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

Identity element:

- $P \wedge \text{True} = P$,
- $P \vee \text{True} = \text{True}$.

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:

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

Conjunctive Normal Form(CNF)

Conjunctive normal form (CNF) is an important normal form for propositional logic. A logic formula is in conjunctive normal form, if it is a single conjunction of disjunctions of (possibly negated) literals. No more nesting and no other negations are allowed.

Hene,

(1) literal to be either a propositional letter or its negation, e.g., b or ¬b.

(2) Clause is a disjunction $\ell_1 \vee \ell_2 \vee \ldots \vee \ell_n$ of one or more literals ℓ_i.

example: - p ∨ ¬r ∨ ¬p

Examples of Conjunctive normal form is: -

- a
- ¬b
- a ∧ b
- (a ∨ ¬b) ∧ (c ∨ d)
- ¬a ∧ (b ∨ ¬c ∨ d) ∧ (a ∨ ¬d)

OR

In conjunctive normal form, statements in Boolean logic are conjunctions of clauses with clauses of disjunctions. In other words, a statement is a series of ORs connected by ANDs.

For example:

- (A OR B) AND (C OR D)
- (A OR B) AND (NOT C OR B)

Any arbitrary formula in propositional logic can be transformed into conjunctive normal form by application of the laws of distribution, De Morgan's laws, and by removing double negations.

To convert to conjunctive normal form, we use the following rules:

1) Apply Double Negation:

- **P↔¬(¬P)**

2) Apply De Morgan's Laws

- **¬(P∨Q)↔(¬P)∧(¬Q)**
- **¬(P∧Q)↔(¬P)∨(¬Q)**

3) Apply Distributive Laws

- **(P∨(Q∧R))↔(P∨Q)∧(P∨R)**
- **(P∧(Q∨R))↔(P∧Q)∨(P∧R)**

OR

Procedure to convert a statement to CNF

1. **Eliminate implications and biconditionals using formulas:**
   - **$(p \leftrightarrow q) \Rightarrow (p \rightarrow q) \wedge (q \rightarrow p)$**
   - **$p \rightarrow q \Rightarrow \neg p \vee q$**
2. **Apply De-Morgan's Law and reduce NOT symbols so to bring negations before the atoms :**
   - **$\neg(P \vee Q) \leftrightarrow (\neg P) \wedge (\neg Q)$**
   - **$\neg(P \wedge Q) \leftrightarrow (\neg P) \vee (\neg Q)$**
3. **Use distributive and other laws and equivalent formulas to obtain Normal forms.**

Example:-

1) Converting to conjunctive normal form:

$\neg((\neg p \rightarrow \neg q) \wedge \neg r) \equiv \neg((\neg \neg p \vee \neg q) \wedge \neg r)$ [definition]
$\equiv \neg((p \vee \neg q) \wedge \neg r)$ [double negation]
$\equiv \neg(p \vee \neg q) \vee \neg\neg r$ [DeMorgan's]
$\equiv \neg(p \vee \neg q) \vee r$ [double negation]
$\equiv (\neg p \wedge \neg\neg q) \vee r$ [DeMorgan's]
$\equiv (\neg p \wedge q) \vee r$ [double negation]
$\equiv (\neg p \vee r) \wedge (q \vee r)$ [distributive]

2) Converting to conjunctive normal form:

$\neg(p \rightarrow q) \vee (r \rightarrow p) \equiv \neg(\neg p \vee q) \vee (\neg r \vee p)$ [definition]
$\equiv (p \wedge \neg q) \vee (\neg r \vee p)$ double negation
$\equiv (p \vee \neg r \vee p) \wedge (\neg q \vee \neg r \vee p)$ associative/distributive law

3) Converting to conjunctive normal form:

$((p \rightarrow q) \rightarrow r) = (\neg p \vee q) \rightarrow r$
$= \neg(\neg p \vee q) \vee r$
$= (p \wedge \neg q) \vee r$
$= (p \vee r) \wedge (\neg q \vee r)$

Disjunctive Normal Form (DNF)

A logical formula is said to be in disjunctive normal form if it is a disjunction of conjunctions with every variable and its negation is present once in each conjunction. All disjunctive normal forms are non-unique, as all disjunctive normal forms for the same proposition are mutually equivalent.

A disjunction of conjunctions where every variable or its negation is represented once in each conjunction (a minterm) , each minterm appears only once

Example:

All of the following formulas are in DNF:

- $(A \wedge \neg B \wedge \neg C) \vee (\neg D \wedge E \wedge F)$
- $(A \wedge B) \vee C$
- $A \wedge B$
- A

However, the following formulas are **not** in DNF:

- $\neg(A \vee B)$, since an OR is nested within a NOT
- $\neg(A \wedge B) \vee C$, since an AND is nested within a NOT
- $A \vee (B \wedge (C \vee D))$, since an OR is nested within an AND

Converting a formula to DNF involves using logical equivalences, such as double negation elimination, De Morgan's laws, and the distributive law. All logical formulas can be converted into an equivalent disjunctive normal form

Example:

1. Converting to disjunctive normal form

$$
\begin{aligned}
(p \to q) \to (\neg r \wedge q) &\equiv \neg(p \to q) \vee (\neg r \wedge q) && \text{[definition]} \\
&\equiv \neg(\neg p \vee q) \vee (\neg r \wedge q) && \text{[definition]} \\
&\equiv (\neg\neg p \wedge \neg q) \vee (\neg r \wedge q) && \text{[DeMorgan's]} \\
&\equiv (p \wedge \neg q) \vee (\neg r \wedge q) && \text{[double negation]}
\end{aligned}
$$

Rule of inference

In logic, inference rule is a logical form consisting of a function which takes premises, analyses their syntax, and returns a conclusion . Or in another words Rules of inference are templates for building valid arguments.

Here …
An **argument** in propositional logic is sequence of propositions or a sequence of statements that end with a conclusion.it includes premises, assumption, hypothesis, conclusion etc.

The last statement is the conclusion and all its preceding statements are called premises (or hypothesis).

An argument is valid if the truth of all its premises implies that the conclusion is true. In other words, an argument form with premises p1, p2, . . . , pn and conclusion q is valid if and only if (p1 ∧ p2 ∧ · · · ∧ pn) → q is a tautology.

The symbol "∴", (read therefore) is placed before the conclusion. A valid argument is one where the conclusion follows from the truth values of the premises. Rules of Inference provide the templates or guidelines for constructing valid arguments from the statements that we already have.

Rule of Inference	Name	Rule of Inference	Name
$\dfrac{P}{\therefore P \vee Q}$	Addition	$\dfrac{P \vee Q \quad \neg P}{\therefore Q}$	Disjunctive Syllogism
$\dfrac{P \quad Q}{\therefore P \wedge Q}$	Conjunction	$\dfrac{P \to Q \quad Q \to R}{\therefore P \to R}$	Hypothetical Syllogism
$\dfrac{P \wedge Q}{\therefore P}$	Simplification	$\dfrac{(P \to Q) \wedge (R \to S) \quad P \vee R}{\therefore Q \vee S}$	Constructive Dilemma

$\begin{array}{l} P \to Q \\ P \\ \hline \therefore Q \end{array}$	Modus Ponens	$\begin{array}{l} (P \to Q) \\ \wedge\, (R \to S) \\ \neg Q \vee \neg S \\ \hline \therefore \neg P \vee \neg R \end{array}$	Destructive Dilemma
$\begin{array}{l} P \to Q \\ \neg Q \\ \hline \therefore \neg P \end{array}$	Modus Tollens		

Addition

If P is a premise, we can use Addition rule to derive P∨Q.

$$\frac{P}{\therefore P \vee Q}$$

Example

Let P be the proposition, "Lisa studies very hard" is true
Therefore − "Either Lisa studies very hard Or She is a very bad student."
Here Q is the proposition "She is a very bad student".

Conjunction

If P and Q are two premises, we can use Conjunction rule to derive P∧Q.

$$\begin{array}{l} P \\ Q \\ \hline \therefore P \wedge Q \end{array}$$

Example

Let P − "She studies very hard"
Let Q − "She is the best girl in the class"

Therefore − "She studies very hard and she is the best girl in the class"

Simplification

If P∧Q is a premise, we can use Simplification rule to derive P.

$$\frac{P \wedge Q}{\therefore P}$$

Example

"She studies very hard and She is the best girl in the class", P∧Q
Therefore − " She studies very hard"

Modus Ponens

If P and P→Q are two premises, we can use Modus Ponens to derive Q.

$$\frac{\begin{array}{c} P \rightarrow Q \\ P \end{array}}{\therefore Q}$$

Example

"if it is raining. Lisa has an umbrella", P→Q
"it is raining", P
Therefore − "Lisa has umbrella "

Modus Tollens

If P→Q and ¬Q are two premises, we can use Modus Tollens to derive ¬P

$$\frac{\begin{array}{c} P \rightarrow Q \\ \neg Q \end{array}}{\therefore \neg P}$$

Example

" if it is raining. Lisa has an umbrella ", P→Q
" Lisa doesn't have an umbrella ", ¬Q
Therefore − "It is not raining "

Disjunctive Syllogism

If ¬P and P∨Q are two premises, we can use Disjunctive Syllogism to derive Q.

$$\frac{\neg P \quad P \vee Q}{\therefore Q}$$

Example

"The ice cream is not vanilla flavored", ¬P
"The ice cream is either vanilla flavored or chocolate flavored", P∨Q
Therefore − "The ice cream is chocolate flavored"

Hypothetical Syllogism

If P→Q and Q→R are two premises, we can use Hypothetical Syllogism to derive P→R

$$\frac{P \to Q \quad Q \to R}{\therefore P \to R}$$

Example

"If it rains, I shall not go to school", P→Q
"If I don't go to school, I won't need to do homework", Q→R
Therefore − "If it rains, I won't need to do homework"

Constructive Dilemma

If (P→Q)∧(R→S) and P∨R are two premises, we can use constructive dilemma to derive Q∨S.

$$\frac{(P \to Q) \wedge (R \to S) \quad P \vee R}{\therefore Q \vee S}$$

Example
"If it rains, I will take a leave", (P→Q)
"If it is hot outside, I will go for a shower", (R→S)
"Either it will rain or it is hot outside", P∨R
Therefore − "I will take a leave or I will go for a shower"

Destructive Dilemma

If (P→Q)∧(R→S) and ¬Q∨¬S are two premises, we can use destructive dilemma to derive ¬P∨¬R.

$$\frac{(P \to Q) \land (R \to S) \quad \neg Q \lor \neg S}{\therefore \neg P \lor \neg R}$$

Example
"If it rains, I will take a leave", (P→Q)
"If it is hot outside, I will go for a shower", (R→S)
"Either I will not take a leave or I will not go for a shower", ¬Q∨¬S
Therefore − "Either it does not rain or it is not hot outside"

Rules of Inference with corresponding tautology are tabulated below –

Rule of Inference	Tautology	Name
$\frac{p \quad p \to q}{\therefore q}$	$(p \land (p \to q)) \to q$	Modus Ponens
$\frac{\neg q \quad p \to q}{\therefore \neg p}$	$(\neg q \land (p \to q)) \to \neg p$	Modus Tollens
$\frac{p \to q \quad q \to r}{\therefore p \to r}$	$((p \to q) \land (q \to r)) \to (p \to r)$	Hypothetical syllogism
$\frac{\neg p \quad p \lor q}{\therefore q}$	$(\neg p \land (p \lor q)) \to q$	Disjunctive Syllogism
$\frac{p}{\therefore (p \lor q)}$	$p \to (p \lor q)$	Addition
$\frac{(p \land q) \to r}{\therefore p \to (q \to r)}$	$((p \land q) \to r) \to (p \to (q \to r))$	Exportation
$\frac{p \lor q \quad \neg p \lor r}{\therefore q \lor r}$	$((p \lor q) \land (\neg p \lor r)) \to q \lor r$	Resolution

Inference rules in proposition Logic

1.Idempotent Rule:

- P∧ P= P
- P ∨ Q = P.

2.Commutative Rule:

- P∧ Q= Q ∧ P
- P ∨ Q = Q ∨ P.

3.Associative Rule:

- P ∧ (Q ∧ R) = (P ∧ Q) ∧ R
- P ∨ (Q ∨ R) = (P ∨ Q) ∨ R

4.Distributive Rule:

- P∧ (Q ∨ R) = (P ∧ Q) ∨ (P ∧ R).
- P ∨ (Q ∧ R) = (P ∨ Q) ∧ (P ∨ R).

5.DE Morgan's Rule:

- ¬ (P ∧ Q) = (¬P) ∨ (¬Q)
- ¬ (P ∨ Q) = (¬ P) ∧ (¬Q).

6.Implication elimination:

- P->Q = ¬P ∨ Q

7.Bidirectional Implication elimination:

- P<->Q = (P->Q) ∧(Q->P)

8.Contrapositive Rule:

- P->Q = ¬P -> ¬Q

9.Doublc-ncgation climination:

- ¬ (¬P) = P.

10.Absorption Rule:

- P ∨ (P ∧ Q) = P
- P ∧ (P ∨ Q) = P

11.Fundamental identities:

- P ∧ ¬P = F [contradiction]
- P ∨ ¬P = T [tautology]

Similarly……..

P∨T=P

P∨F=P

P∨ ¬T=P

P∧F =F
P∧T=P

12. Modus Ponens:

If **P** is true and **P→Q** then we can infer **Q** is also true.

P
P→Q

Hence, Q

13. Modus Tollens:

If **¬P** is true and **P→Q** then we can infer **¬Q** .

¬P
P→Q

Hence, ¬Q

14. Chain rule:

If p→q and q→r then p→r

15. Disjunctive Syllogism:

if **¬p** and **p∨q** we can infer **q** is true.

16. AND elimination:

Given P and Q are true then we can deduce P and Q seperately: $P \wedge Q \rightarrow P$

$P \wedge Q \rightarrow Q$

17. AND introduction:

Given **P** and **Q** are true then we deduce **P ∧ Q**

18. OR introduction:

Given P and Q are true then we can deduce P and Q separately:

$$P \rightarrow P \vee Q$$
$$Q \rightarrow P \vee Q$$

Inference Using Resolution: -

In propositional logic, resolution method is the only inference rule which gives a new clause when two or more clauses are coupled together.

The resolution rule in propositional logic, which is a single valid inference rule that produces a new clause implied by two clauses containing complementary literals. A literal is a propositional variable or the negation of a propositional variable. Two literals are said to be complements if one is the negation of the other. The resulting clause contains all the literals that do not have complements. Formally:

$$\frac{a_1 \vee a_2 \vee \cdots \vee c, \quad b_1 \vee b_2 \vee \cdots \vee \neg c}{a_1 \vee a_2 \vee \cdots \vee b_1 \vee b_2 \vee \cdots}$$

where

all a_i, b_i, and c are literals,

the dividing line stands for "entails".

The above may also be written as:

$$\frac{(\neg a_1 \wedge \neg a_2 \wedge \cdots) \rightarrow c, \quad c \rightarrow (b_1 \vee b_2 \vee \cdots)}{(\neg a_1 \wedge \neg a_2 \wedge \cdots) \rightarrow (b_1 \vee b_2 \vee \cdots)}$$

The clause produced by the resolution rule is called the resolvent of the two input clauses.

When the two clauses contain more than one pair of complementary literals, the resolution rule can be applied (independently) for each such pair; however, the result is always a tautology. Modus ponens can be seen as a special case of resolution (of a one-literal clause and a two-literal clause).

$$\frac{p \rightarrow q, \quad p}{q}$$

is equivalent to

$$\frac{\neg p \vee q, \quad p}{q}$$

Resolution method is also called **Proof by Refutation**. Since the knowledge base itself is consistent, the contradiction must be introduced by a negated goal. As a result, we have to conclude that the original goal is true.

The process followed to convert the propositional logic into resolution method contains the following steps:

- Convert the given axiom into clausal form, i.e., disjunction form.
- Apply and proof the given goal using negation rule.
- Use those literals which are needed to prove.
- Solve the clauses together and achieve the goal.

Example OF Propositional Resolution……

Consider the following Knowledge Base:

1. The humidity is high or the sky is cloudy.
2. If the sky is cloudy, then it will rain.
3. If the humidity is high, then it is hot.
4. It is not hot.

Goal: It will rain.

Use propositional logic and apply resolution method to prove that the goal is derivable from the given knowledge base.

Solution: Let's construct propositions of the given sentences one by one:

1) Let, P: Humidity is high.
   Q: Sky is cloudy.

   It will be represented as P V Q.

2) Q: Sky is cloudy. …from (1)
   Let, R: It will rain.

It will be represented as Q → R.

3) P: Humidity is high. …from (1)
   Let, S: It is hot.
   It will be represented as P → S.

4) ¬S: It is not hot.

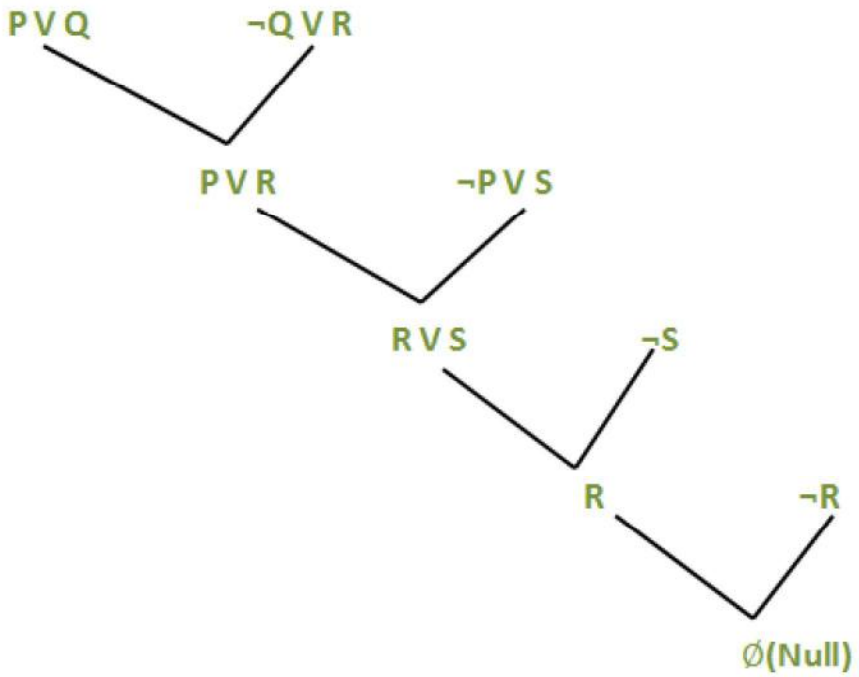
Applying resolution method:

In (2), Q → R will be converted as (¬Q V R)

In (3), P → S will be converted as (¬P V S)

Negation of Goal (¬R): It will not rain.

Finally, apply the rule as shown below:



After applying Proof by Refutation (Contradiction) on the goal, the problem is solved, and it has terminated with a Null clause (Ø). Hence, the goal is achieved. Thus, It is not raining.

Backward Chaining and Forward Chaining

Forward Chaining and Backward Chaining are the two most important strategies in the field of Artificial Intelligence and lie in the Expert System Domain of AI. Forward and Backward chaining is the strategies used by the Inference Engine in making the deductions. So, before understanding forward and backward chaining lets first understand that from where these two terms came.

Inference Engine:

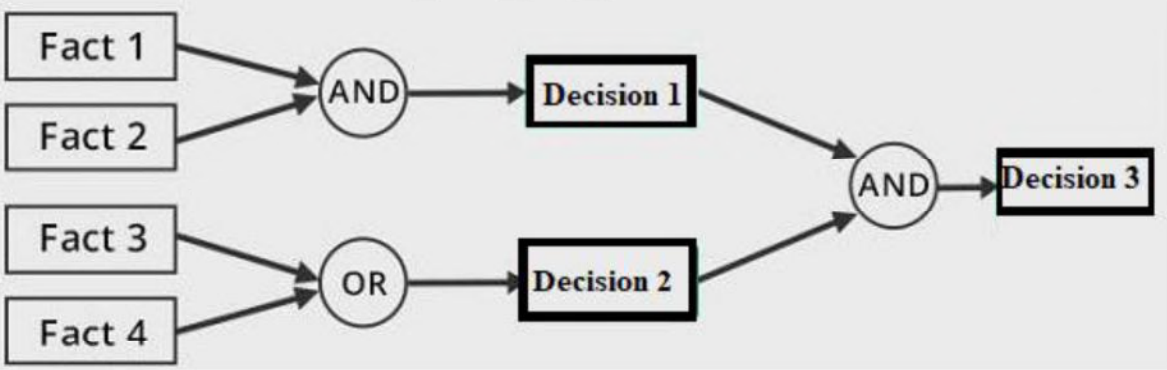
It is the component of the intelligent system/Expert system in AI that applies logical rules to the knowledge base to deduce new inference. A knowledgebase is a structured collection of facts about the system's domain. The inference engine works on two modes…

1) Forward chaining
2) Backward changing

Forward Chaining

Forward Chaining is the process which works on the basis of available data to make certain decisions. Forward chaining is the process of chaining data in the forward direction. In forward chaining, we start with the available data and use inference rules to extract data until the goal is not reached. Forward chaining is the concept of data and decision. From the available data, expand until a decision is not made.

In forward 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 in Propositional Logic

In propositional logic, forward chaining starts its journey from the given knowledge base. If all the premises of the implication are known, then its conclusion will be added to the set of known facts.

Properties of Forward Chaining

- It starts with known facts and asserts the new facts.
- It follows a bottom-up approach i.e. the reasoning deduction moves from bottom to the top
- It is also called a Data-driven approach as it relies on existing data to reach the goal state
- It is conclusion driven i.e. its objective is to reach the conclusion from the initial state
- It is widely used in the Expert System like CLIPS and Production rule system

Advantages of Forward Chaining

- Forward Chaining works great when the available information can be used to reach the goal state
- Forward Chaining has the ability to provide lots of data from the limited initial data
- Forward Chaining is best suited for Expert system application that requires more control, planning, and monitoring
- Forward Chaining should be applied when there are a limited number of initial states or facts

Disadvantages of Forward Chaining

- The inference engine will generate new information without knowing which information will be relevant in reaching the goal state
- The user might have to enter a lot of information initially without knowing which information will be used to reach the goal state
- Inference Engine may fire many rules which don't contribute toward reaching the goal state
- It might give different conclusion which may result in the high cost of the chaining process

Example-1:

Let's look at an example to understand how Forward Chaining works in practice
Rule 1: IF A is human THEN A is mammal
Rule 2: IF A is a mammal, THEN A is a living form
Rule 3: IF A is a living form, THEN A is mortal

Fact: Shyam is human

From these inference rules, we have to reach the Goal
Goal: Is Shyam a mortal?

Steps:

1. Start with the Known fact. We know that Shyam is human (From the Fact statement).
2. Using R1 we can infer that Shyam is a mammal. Since it is not a Goal Statement so continue.
3. Then jump to Rule 2: if Shyam is a mammal, then it as a living form so we can say that mortal is a living form. Since it is not a Goal Statement so continue

4. Using R3, Since Shyam is a life form so it must be Mortal. Since it is the goal statement so Exit

Example-2: -

1. If D barks and D eats bone, then D is a dog.
2. If V is cold and V is sweet, then V is ice-cream.
3. If D is a dog, then D is black.
4. If V is ice-cream, then it is Vanilla.

Fact:

- Tomy barks.
- Tomy eats bone.

Goal: Tomy is black

Solution: Given Tomy barks.
From (1), it is clear: If Tomy barks and Tomy eats bone, then Tomy is a dog.
From (3), it is clear: If Tomy is a dog, then Tomy is black.

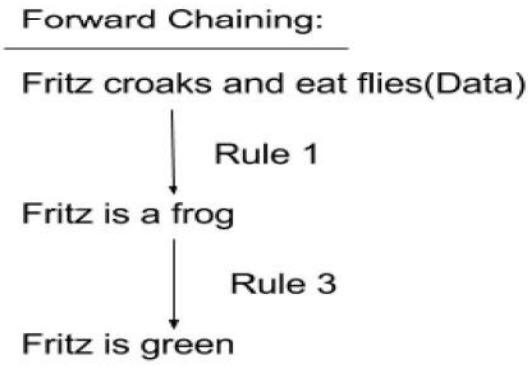
Hence, it is proved that Tomy is black.

Example-3: consider the rule base as:

I) If x croaks and eats flies, then x is a frog.
II) If x chirps and sings, then x is a canary.
III) If x is a frog, then x is green
IV) If x is a canary, then x is yellow.

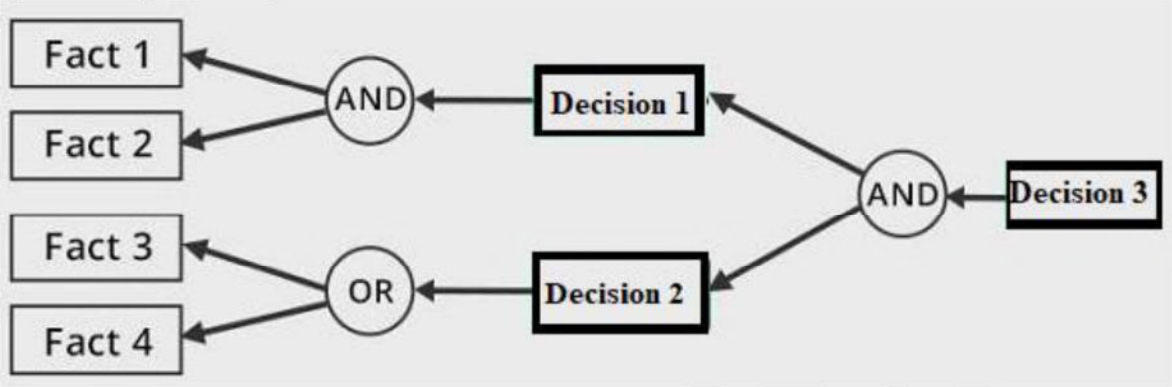
Conclude colour of fritz, given the croaks and eats flies.

Ans: -



Backward Chaining

Backward Chaining is a backward approach which works in the backward direction. It begins its journey from the back of the goal. In another words it starts with goals and works backword to determine what facts must be asserted so that the goals can be achieved. Like, forward chaining, we have backward chaining for Propositional logic as well as Predicate logic followed by their respective algorithms.



Backward Chaining in Propositional Logic

In propositional logic, backward chaining begins from the goal and using the given propositions, it proves the asked goal.

How Backward Propagation works?

Backward Chaining or Backward Propagation is the reverse of Forward Chaining. It starts from the goal state and propagates backwards using inference rules so as to find out the facts that can support the goal. It is also called as Goal-Driven reasoning.

It starts from the given goal, searches for the THEN part of the rule (action part) if the rule is found and its IF part matches the Inference Rule, then the rule is executed other Inference Engine set it as a new subgoal.

Rule 1: IF A AND B THEN C
Rule 2: IF C THEN E
Rule 3: IF A AND E THEN H
Facts: A, B
Goal: Prove H

Proof:

Step 1: At first system looks for the statement that has goal on the R.H.S i.e., R3 then look for the L.H.S of the rule to check if it contains the fact. It contains A and E but we need B also
Step 2: Now it will have E as the sub goal which is proved by Rule 2. Now look at its L.H.S i.e. C
Step 3: C can be proved by Rule 1 which has A & B as the L.H.S
Step 4: Since we got both the facts A&B from the goal so the Algorithm end here
Step 5: Stop

Properties of Backward Chaining

- Backward Chaining is a top-down approach where we start from the goal state and works backwards to find the required facts that support the goal statement
- It is known as Goal-driven approach as we start from the goal and then divide into sub-goal to extract the facts
- It applies the Depth-First search strategy
- It can only generate a limited number of conclusions
- It only tests for few of the required rules

Advantages of Backward Chaining

- The search in backward chaining is directed so the processing terminates when the fact is verified
- Backward Chaining consider only relevant parts of knowledge base so it never performs unnecessary inferences
- Unlike Forward Chaining, here only a few data points are needed but rules are searched exhaustively
- It is very efficient for problems like diagnosing and debugging

Disadvantages

- Since backward chaining is goal-driven, so the goal must be known beforehand to perform backward chaining
- It is difficult to implement backward chaining

Example-1:

Given that:

1. If D barks and D eats bone, then D is a dog.
2. If V is cold and V is sweet, then V is ice-cream.
3. If D is a dog, then D is black.
4. If V is ice-cream, then it is Vanilla.

Derive backward chaining using the given known facts to prove Tomy is black.

- Tomy barks.
- Tomy eats bone.

Solution:

1.On replacing D with Tomy in (3), it becomes:

If Tomy is a dog, then Tomy is black.

Thus, the goal is matched with the above axiom.

- Now, we have to prove Tomy is a dog. …(new goal)

Replace D with Tomy in (1), it will become:

If Tomy barks and Tomy eats bone, then Tomy is a dog. …(new goal)

Again, the goal is achieved.

- Now, we have to prove that Tomy barks and Tomy eats bone. …(new goal)

As we can see, the goal is a combination of two sentences which can be further divided as:

Tomy barks.
Tomy eats bone.
From (1), it is clear that Tomy is a dog.
Hence, Tomy is black.

Note: Statement (2) and (4) are not used in proving the given axiom. So, it is clear that goal never matches the negated versions of the axioms. Always Modus Ponen is used, rather Modus Tollen.

Example-2: consider the rule base as:

I) If x croaks and eats flies, then x is a frog.
II) If x chirps and sings, then x is a canary.
III) If x is a frog, then x is green
IV) If x is a canary, then x is yellow.

Conclude colour of fritz, given the croaks and eats flies.

Ans: -

Comparison of Forward and Backward Chaining

The concept of comparison can be understood with the help of an example of day-to-day life. Every day, due to a lot of reasons we have to visit hospitals for medical consultation. What logic is applied for the treatment of a person? There are two approaches can be used by the physician. One method is of forward chaining process and second one is of backward chaining method.

A physician usually begins to diagnose a patient by asking him about the symptoms he or she suffers from, such as high blood pressure, temperature, headache, sore throat, coughing and many other symptoms. After the analysis and discussion, information is collected for the starting

process of treatment. The physician uses this information to draw a reasonable conclusion or to establish a hypothesis to explore further. This way of reasoning is done in a knowledge base system which is called forward-chaining.

On the contrast, a physician may suspect some problems with patient. After check-up, he attempts to prove by looking for certain symptoms of the disease which illustrates the concept of backward chaining. Table below shows the comparative study of chaining methods of artificial intelligence.

Forward Chaining	**Backward Chaining**
It starts with new data.	It starts with some goal or hypothesis.
It asks few questions.	It asks many questions.
It examines all rules.	It examines some rules.
Slow approach.	Fast approach.
Gather larger information from small amount of data.	It produce small amount of information from available data.
Forward Chaining is primarily data driven.	Backward Chaining is primarily Goal Driven.
It uses its input. It searches rules for answers.	It proves the considered hypothesis.
It is buttom up approach	**It is top down approach**
Works forward to find conclusions from facts.	Works backward to find facts that support the hypothesis.
It tends to breath – first.	It tends to depth – first.
Forward Chaining is suitable for problems that start from data collection; e.g. planning, monitoring and control.	Backward Chaining is suitable for problems that start from hypothesis, e.g. diagnosis.
This type of chaining is non-focused because it infers all conclusions, may answer unrelated questions.	This type of chaining is focused to prove the goal and search as only the part of knowledge base that is related to the problem.
Explanation is not facilitated in Forward Chaining.	Explanation is facilitated in Backward Chaining
All data is available.	Data must be acquired interactively (i.e. on demand)
It deals with less number of initial states and many results.	It deals with less starting goals and many facts.
Forming a goal is difficult in case of Forward Chaining.	Forming a goal is easy in case of Backward Chaining.

Predicate Logic

Predicate logic is an extension of Propositional logic. It adds the concept of predicates and quantifiers to better capture the meaning of statements that cannot be adequately expressed by propositional logic.

Predicates are functions that map variables to truth values. They are essentially Boolean functions whose value could be true or false, depending on the arguments to the predicate. They are generalizations of propositional variables. A propositional variable is a predicate with no arguments.

So, A predicate is a statement that contains variables (predicate variables), and they may be true or false depending on the values of these variables.

Example:

P(x) = "x^2 is greater than x" is a predicate. It contains one predicate variable **x**. If we choose **x = 1**, P (1) is "1 is greater than 1", which is a proposition (always false).

Since a predicate takes value true or false once instantiated (that is, once its variables are taking values), we may alternatively say that a predicate instantiated becomes a proposition. Predicate logic also known as First order logic or first order predicate logic.

First-Order Predicate Logic (FOPL)

In propositional logic, we can only represent the facts, which are either true or false. PL is not sufficient to represent the complex sentences or natural language statements. The propositional logic has very limited expressive power. 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 called first-order predicate logic.

first-order predicate logic.

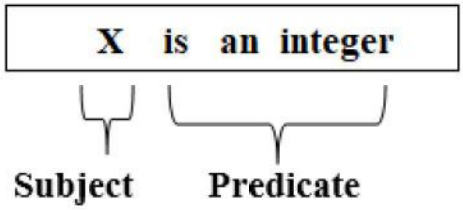
- first-order predicate logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.
- first-order predicate logic is sufficiently expressive to represent the natural language statements in a concise way.
- first-order predicate logic is a powerful language that develops information about the objects in an easier way and can also express the relationship between those objects.
- first-order predicate 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, colours, 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, ...

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.



First-order predicate logic consists of two sentences they are…

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.

Syntax and semantics in FOPL

- Like propositional logic syntax for FOPL is determine by the allowable symbols and rules of combination.
- In prepositional logic, every expression is a sentence that represents a fact.
- First order logic includes the sentences along with terms which can represent the objects.
- Constant symbols, variables and function symbols are used to build terms, while quantifiers and predicate symbols are used to build the sentences.

Syntax:

Constant	A, B, C.....
Functions	Size, Color
Variables	x, a
Terms	Constant, variable or function(Term..)
Predicates	True, False
Quantifiers	∀, ∃
Atomic Sentences	Predicate, Predicate(Term,...), Term=Term
Sentences	¬ Sentence, Sentence ∨ Sentence, Sentence ∧ Sentence, Sentence ⇒ Sentence, Sentence ⇔ Sentence, Quantifier Variable,... Sentence

Which can be further explained as:

1. **Domain:** a non-empty set of objects
2. **Constant:**
   Constants are used to denote objects. The objects can be anything. A single object can be denoted by multiple individual constants, reflecting the fact that objects can have multiple names. In FOPL constant are like 1, 2, A, John,kathmandu, cat, etc…
3. **Variables:** Variables are also used to denote objects, but not a specific one. That is, a variable can be used to denote some arbitrary object, just like variables in mathematics **Example:** x, y, z, a, b....
4. **Predicates:** Predicates take an arbitrary but finite number of arguments. It takes objects in the domain as arguments and returns true or false. They describe properties of objects or relationships between objects. **Example**: Brother, Father, >,....
   The number of arguments is called the arity of that function and predicate.

5. **Function:** Functions denotes relation defined on a domain. Example sqrt, LeftLegOf, ...
6. **Connectives:** Connectives can be used to recursively build complex formulas, just like propositional logic. Example ∧, ∨, ¬, ⇒, ⇔
7. **Equality** ==
8. **Quantifier**: ∀, ∃

Semantics

The semantics of FOPL are determined by interpretation assigned to predicates, rather than propositions. Let's understand with an example, Consider the sentence "Elephants are big".
This can be expressed as …

HasSize(Elephant, Big)

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 symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifiers:
    1. Universal Quantifier (∀), (for all, everyone, everything)
    2. 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.

∀x man(x) → drink (x, coffee).

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

All kings are persons

∀x King(x) ⇒ Person(x) .

"For all x, if x is a king, then x is a person."

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.

$\exists x$: boys(x) $\wedge$ intelligent(x)

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

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.

$\forall x$ bird(x) $\rightarrow$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 $\forall$, and it will be represented as follows:

$\forall x$ man(x) parent(y) $\rightarrow$ respects (x, y).

$\forall x$ man(x) $\rightarrow$ 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 $\exists$, and it will be represented as:

$\exists x$: boys(x) $\wedge$ 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 $\forall$ with negation, so following representation for this:

$\neg\forall$ (x) [student(x) $\rightarrow$ like(x, Mathematics) $\wedge$ like(x, Science)].

Other Example:-

1. Marcus is a man.
   Man(Marcus)

2. Marcus was a Pompein
   Pompein (Marcus)

3. All Pompeins were Romans
   $\forall x$: Pompeins(x) $\rightarrow$Romans(x)

4. Every gardener likes sun
   $\forall x$: gardener(x) $\rightarrow$likes(x,sun)

5. All purple mushrooms are poisonous
   $\forall x$: Mushroom(x) $\wedge$ Purple(x)$\rightarrow$Poisonous(x)

6. Everyone is loyal to someone

$\forall x \exists y$: Loyal(x,y)

7. Everyone loves everyone
   $\forall x \forall y$: Loves(x,y)

9. Everyone loves everyone except himself
   $\forall x \forall y$: Loves(x,y) $\wedge \neg$Loves(x,x)

10. All Romans either loyal to ceaser or hated him
    $\forall x$: Romans(x) $\rightarrow$Loyal(x, ceaser)v Hated(x,ceasar)

11. People only try to assassinate rulers they are not loyal to
    $\forall x \forall y$: person(x) $\wedge$ ruler(y) $\wedge$ tryassasinate(x,y)=> $\neg$loyal(x,y)

12. Anyone who is married and has more than one spouse is a Bigamist
    $\forall x$: married(x) $\wedge$ (no.of. spouse(x) > 1) => bigamist(x)

13. You can fool all the people some of the time
    $\forall x \exists y$: person(x) => time(y) $\wedge$ canfool(x,y)

14. You can fool some of the people all the time
    $\exists x \forall t$: person(x) $\wedge$time(t) => canfool(x,t)

15. Every city has a dog catcher who has been bitten by every dog in the town.
    $\forall x \exists y \forall z$: city(x) => dogcatcher(y) $\wedge$ dog(z) $\wedge$ lives(z,x) $\wedge$ bytes(z,y)

16. Ones husband is ones male spouse
    $\forall w \exists h$: husband(h,w) => male(y) $\wedge$ spouse(h,w)

17. Parents and children are inverse relation
    $\forall p \forall c$: parent(p,c) => child(c,p)

18. A sibling is another child of ones parent
    $\forall x \forall y \exists p$: sibling(x,y) => x=/=y $\wedge$ parent(p,x) $\wedge$ parent(p,y)

19. The best score in maths is always higher than the best score in English.
    $\forall x \forall y$: bestmathscore(x) $\wedge$ bestenglishscore(y) => x>y

20. There is a barber who shaves all men in town who do not shave themselves.
    $\forall x \forall y \exists z$: man(x) $\wedge$ town(y) $\wedge$ lives(x,y) $\wedge \neg$shaves(x,x) => barber(z) $\wedge$ shaves(z,x)

Inference with FOPL: By converting into PL (Existential and universal instantiation)

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

1. Universal Generalization:

- Universal generalization is a logical inference rule in first-order predicate logic (FOPL) that allows you to conclude that a statement is true for all members of a particular domain based on the fact that it is true for some arbitrary individual within that domain.
- That is, 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:

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

- In this rule, the variable x represents any individual in the domain, and P(x) is a statement that applies to that individual. The symbol ∀ (for all) indicates that P(x) holds for all individuals in the domain.
- The rule of universal generalization can be used to prove statements about all members of a domain by establishing the truth of the statement for an arbitrary individual within that domain.

- For example,

  If you can prove that "Ram is tall"…
  Which can be written as

  (P(Ram))

  Then you can apply the rule of universal generalization to conclude that…
  "All people are tall"…
  i.e. ∀xP(x).

  Hence…

  It is important to note that the rule of universal generalization can only be applied to a statement that is true for every individual in the domain. If you can only establish the truth of a statement for some individuals in the domain, then you cannot use universal generalization to conclude that it is true for all individuals.

Example:

Other example:

- P(c): "A byte contains 8 bits",
- so, for ∀ x P(x) "All bytes contain 8 bits.", it will also be true.

2. Universal Instantiation:

- Universal instantiation(UI) is also called as universal elimination. It is a logical inference rule in first-order predicate logic (FOPL) that allows you to instantiate a universally quantified statement by substituting a particular variable with a specific object in the domain.
- It states 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:

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

Here, the variable x represents any individual in the domain, and P(x) is a statement that applies to that individual. The symbol ∀ (for all) indicates that P(x) holds for all individuals in the domain.

Therefore, P(a) holds for a particular object a in the domain.

- It is important to note that universal instantiation must be used carefully, as it can only be applied to a universally quantified statement that has been proven to be true.

Hence,

The rule of universal instantiation allows you to derive a specific statement that is true for a particular object a in the domain.

For example,

If you know that "All dogs are mammals"…

It can be written as…

$\forall x\ dog(x) \rightarrow Mammal(x)$

You can use the rule of universal instantiation to derive the statement …

"Jayantilaal is a mammal"

It can be written as…

Mammal(Jayantilaal)

if you know that Jayantilaal is a dog ….. Dog(Jayantilaal).

Example:1.
IF "Every person like ice-cream"=> ∀x P(x) so we can infer that
"John likes ice-cream" => P(c)

Example:2

For example, suppose we have the sentence:

∀x (boy(x) → play(x, cricket))

This sentence states that for any object x, if x is a boy, then x plays cricket. We can apply Universal Instantiation to this sentence and derive the following:

boy(j) → play(j, cricket)

Where j is any term in the domain of discourse that refers to a boy.

Universal Instantiation is a powerful rule of inference that allows us to derive new sentences from universally quantified sentences, which in turn allows us to reason more effectively about the properties and relationships between objects in a logical system.

Example: 3.
"All kings who are greedy are Evil."

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)),

Example: 4.

$$\forall x\ Likes(x, IceCream)$$

$$\downarrow$$

$$Likes(Ben, IceCream)$$

Example: 5

consider the following formula in FOPL:

$\forall x$ (man(x) $\rightarrow$ mortal(x))

This formula expresses the idea that all men are mortal. If we want to derive the statement that Socrates is mortal, we can use UI as follows:

- $\forall x$ (man(x) $\rightarrow$ mortal(x)) (premise)
  man(Socrates) (premise)
- man(Socrates) $\rightarrow$ mortal(Socrates) (UI: replace x with Socrates)
  mortal(Socrates) (Modus Ponens: conclude that Socrates is mortal)

Here, we substitute the term "Socrates" for the universally quantified variable x in the first formula to obtain the conditional statement "man(Socrates) $\rightarrow$ mortal(Socrates)". Then, we use Modus Ponens to derive the conclusion that Socrates is mortal, based on the premise that Socrates is a man.

3. Existential Instantiation:

- Existential instantiation is also called as Existential Elimination, which is a valid inference rule in first-order logic.
- This rule states that one can infer P(c) from the formula given in the form of $\exists x$ P(x) for a new constant symbol c.
- It can be represented as:

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

- In EI, the variable is substituted by a single new constant symbol.

OR

- Existential Instantiation (EI) is a valid inference rule in first-order predicate logic (FOPL) that allows you to derive a new formula by introducing an existential quantifier and replacing the quantified variable with a term.
  The rule can be stated as follows:

  -> If $\exists x$ P(x) is a formula, then we can infer that P(c) is true, where c is a new constant or variable that does not appear elsewhere in the proof and can be substituted for x in the formula.
- In other words, if we know that there exists at least one object in the domain of discourse that satisfies the predicate P, then we can introduce a new term to represent that object and replace the existentially quantified variable x with that term.

Example:1

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.

In above example the variable be x which is replaced by a constant symbol k for any sentence. The value of k is unique as it does not appear for any other sentence in the knowledge base. Such type of constant symbols is known as Skolem constant. As a result, EI is a special case of Skolemization process.

Example:1

consider the following formula in FOPL:

∃x (man(x) ∧ loves(x, Mary))

This formula expresses the idea that there is at least one man who loves Mary. If we want to derive a statement about this man, we can use EI as follows:

∃x (man(x) ∧ loves(x, Mary)) (premise)

man(c) ∧ loves(c, Mary) (EI: introduce new constant c and replace x with c)

Here, we introduce a new constant c to represent the man who loves Mary, and replace the existentially quantified variable x with c. The resulting formula asserts that c is a man who loves Mary. We can then use this formula to derive further statements or conclusions in the proof. Note that the choice of constant c is arbitrary, and any other constant or variable that does not appear elsewhere in the proof could also be used.

4. Existential introduction/ Existential generalization

- 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:

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

Example:

Let's say that,

"Priyanka got good marks in English."

"Therefore, someone got good marks in English."

Unification and lifting

Unification

- Unification is a process of making two different logical atomic expressions identical by finding a substitution. Unification depends on the substitution process.
- Unification is a kind of binding logic between two or more variables.
- Unification Algorithms is used to unite or combine two identical statements.
- The goal of unification is to make two expressions look like identical by using substation
- It means the meaning of the sentence should not be changed, but it should be expressed in multiple ways.
- The UNIFY algorithm in unification takes two sentences as input and then returns a unifier if one exists:
  - Substation means replacing one variable with another term.
  - It takes two literals as input and makes them identical using substitution.
  - It returns fail if the expressions do not match with each other.

- Let p and q be two atomic sentences and θ be a unifier such that, pθ = qθ, then it can be expressed as UNIFY (p, q).

UNIFY(p,q)=θ where SUBST(θ,p)=SUBST(θ,q).

This is the unification algorithm (UNIFY) which is applying to two predicates p and q and θ is the substituents function. Now you substitute the **θ** that is substituent function on the predicates p and same substituent function θ is apply to the predicate q, after applying the substituent function on both predicate p and q they become identical.

If they become identical then it can be said that they are unifiable otherwise they are not unifiable.

Example: -1

- Let's say there are two different expressions
  P (x, y) and p (a, f(z))
- We need to make both above statements identical to each other.
- Perform the substation.
  P(x,y).....(i)
  P(a,f(z))……(ii)
- Substitute x with a and y with f(z) in the first expression, and it will be represented as a/x and f(z)/y
- With both the substation, the first expression will be identical to the second expression and the substation set will be: [a/x,f(z)/y].

Example: -2

- **Given:** knows (Ram, x). is a predicate
- Here the question is … Whom does ram knows?

- The UNIFY algorithm will search all the related sentence in the knowledge base, which could unify with knows (Ram, x).
  UNIFY (Knows (Ram, x), knows (Ram, Shyam)) ≡{x/Shyam}
  UNIFY (Knows {Ram, x}, knows {y, Akash}) ≡ {x/Akash, y/ram}
  UNIFY (Knows {Ram, x}, knows {x, Raman)) ≡ Fails. Unifier is empty

- The last one failed because we have used the same variable for two persons or the two sentences happens to use the same variable name, x
- Unifications are attempted only with sentences that have some chance of unifying.
- For example, there is no point in trying to
  Unify knows (Ram, x) with brother (Laxman, Ram).

Conditions for unification

- **Predicate symbol must be same…**
  An atoms or expression with different predicate symbol can never be unified.
  tryassissinate (Marcus, Caesar)
  Hate (Marcus, Caeser)
- **Number of Arguments in both expressions must be identical.**
  hate (Marcus)
  hate (Marcus, Caesar)
- **Unification will fail** if there are two similar variables present in the same expression.

Lifting

In predicate logic, lifting is the process of extending a logical statement from a single element to a larger set of elements. This is done by replacing specific elements in the statement with variables that can range over a larger set of elements. It is used in artificial intelligence to reason about properties and relations that hold over sets of objects rather than just individual objects.

For example:- 1

Consider the statement …

"John is a man".

In lifting, we would replace the specific name "John" with a variable, say "x", and the statement would become…

"x is a man".

This lifted statement can now be used to make statements about any man, not just John.

Hence, It is important in artificial intelligence because it allows us to reason about general properties of sets of objects, rather than just specific instances. For example, instead of reasoning

about a specific bird, we can reason about all birds. This makes it possible to develop general rules and theories that apply to many different situations.

Lifting is an important technique in many areas of artificial intelligence, including natural language processing, knowledge representation, and automated reasoning. It allows us to reason about complex properties and relations over sets of objects, which is essential for many tasks in AI.

Lifting is also useful for automated reasoning, as it allows logical statements to be applied to a wide range of situations. This is particularly important in areas such as natural language processing, where the same concepts may be expressed in many different ways. By using lifted statements, we can apply a single rule to many different variations of the same concept.

For example:- 2

Consider the formula…
"all cats are mammals"

It can be expressed in predicate logic as

"$\forall x$ (cat(x) $\rightarrow$ mammal(x))".

If we want to reason about a larger domain that includes not just individual cats but also groups of cats, we can lift this formula to a new formula that talks about the properties of those groups. This can be done by replacing the individual variable "x" with a set variable "X", resulting in the formula…

"$\forall X$ ($\forall x$ ($x \in X \wedge$ cat(x) $\rightarrow$ mammal(x)))",

which expresses the property that any group of cats is a group of mammals.

Lifting can also be used to reason about relations between sets of objects. For example, the formula "all cats like milk" can be lifted to "$\forall X$ ($\forall x$ ($x \in X \wedge$ cat(x) $\rightarrow$ likes_milk(x)))", which expresses the property that any group of cats likes milk.

Example: GMP(Generalised modus ponent) is a lifted version of MP

Generalized modus ponens (GMP) is a logical inference rule that extends the modus ponens (MP) rule to apply to predicate logic statements that contain variables. GMP is a lifted version of MP because it operates on lifted or quantified variables, rather than on specific instances of variables.

The modus ponens (MP) rule is a fundamental inference rule in propositional logic that states that if you have a conditional statement ($p \rightarrow q$) and you also have the antecedent (p), then you can infer the consequent (q).

For example:

$p \rightarrow q$
p
Therefore, q

In predicate logic, variables are used to represent general categories of objects, rather than specific objects.

For example,
The statement "All dogs are mammals" can be represented in predicate logic as…
$\forall x Dog(x) \rightarrow Mammal(x)$,

Where x is a variable that can be replaced with any specific instance of a dog.
Generalized modus ponens extends the modus ponens rule to predicate logic statements that contain variables.

The rule of GMP can be expressed as follows:
$\forall x P(x) \rightarrow Q(x)$
P(a)
Therefore, Q(a)

In this rule, the variable x represents any individual in the domain, and P(x) and Q(x) are statements that apply to that individual. The symbol $\forall$ (for all) indicates that $P(x) \rightarrow Q(x)$ holds for all individuals in the domain.

Hence, GMP allows you to infer a specific instance of a predicate logic statement from a universally quantified statement and a specific instance of one of its components.

For example, if you know that "All dogs are mammals" ($\forall x(Dog(x) \rightarrow Mammal(x))$) and you know that Fido is a dog (Dog(Fido)), then you can use GMP to infer that "Fido is a mammal" (Mammal(Fido)).

Therefore, GMP is a lifted version of MP because it extends the MP rule to predicate logic statements that contain variables, allowing you to infer a specific instance of a statement from a universally quantified statement and a specific instance of one of its components.

Other example:
$\forall x,y,z\ ((parent(x,y) \wedge parent(y,z)) \Rightarrow grandparent(x,z))$
parent(james, john), parent(james, richard), parent(harry, james)
We can derive:

- Grandparent(harry, john), bindings:
  {x/harry,y/james,z/john}
- Grandparent(harry, richard), bindings:
  {x/harry,y/james,z/richard}

Inference using resolution

Resolution in the Predicate Logic

Resolution provides a method for finding contradictions in a database of clauses. Resolution refutation proves a theorem by negating the statement that needs to be proved and then adding this negation to the set of axioms that are known (have been assumed) to be true.

In other words,…

Resolution is an inference rule that produces a new clause from two clauses with complementary literals (p and ¬p).

$$\frac{p \vee A \qquad \neg p \vee B}{A \vee B} \quad \textit{resolution}$$

Hence, From clauses p ∨ A and ¬p ∨ B we derive clause A ∨ B

Here, New clause = resolvent of the two clauses with respect to p

Resolution refutation proofs involve the following steps:

1. Put the premises into clause form.
2. Add the negation of what is to be proved (i.e., negate the goal), in clause form, to the set of premises.
3. Resolve these clauses together, producing new clauses that logically follow from them.
4. Produce a contradiction by generating what is referred to as the empty clause.
5. The substitutions used to produce the empty clause are precisely those under which the opposite of the negated goal is true.

Resolution is refutation complete. This means that a contradiction can always be generated whenever one exists. Resolution refutation proofs require that the premises and the negation of the conclusion be placed in a normal form called clause form (as was required in the propositional logic). Clause form represents both the premises and the negation of the conclusion as a set of disjunctions of literals.

Steps for Resolution:

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

To better understand all the above steps, we will take an example in which we will apply resolution.

We use the following well-known argument to illustrate resolution proofs.

Premise 1) Socrates is a mortal.
Premise 2) All mortals will die.

Conclusion: Socrates will die.

First, we **represent this argument in the predicate logic**. We use the predicates Mortal(x) and Will Die (x).

Premise 1) Mortal (Socrates)
Premise 2) (∀x) (Mortal (x) ⇒ Will _ Die (x))

Conclusion) ∴ Will _ Die (Socrates).

Note: The parentheses around Mortal(x) Will _ Die (x) are not necessary; but they aid clarity

Next, the **premises are converted into clause form**:

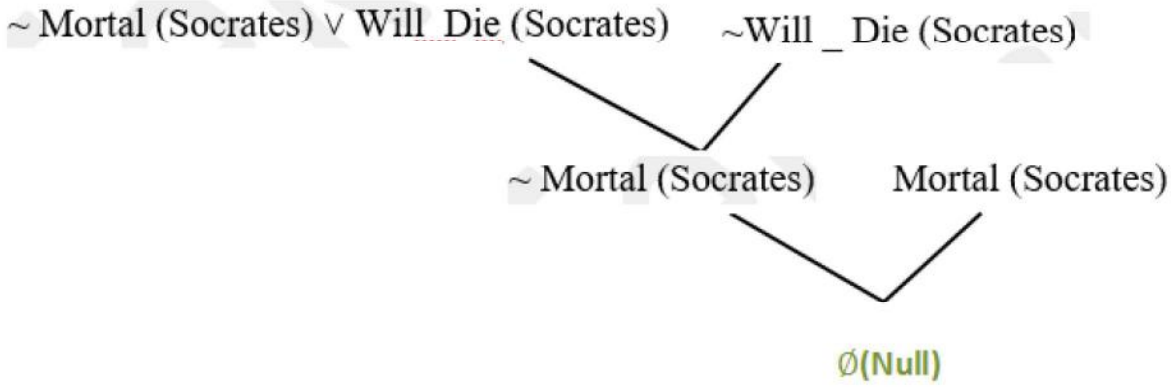
Premise 1) Mortal (Socrates)
Premise 2) ~ Mortal (x) ∨ Will _ Die (x)

Negate the conclusion: ~ Will _ Die (Socrates)
Observe this last predicate is already in clause form.
Our clause base consists of:

1) Mortal (Socrates)
2) ~ Mortal (x) ∨ Will_Die (x)
3) ~Will _ Die (Socrates)

Combining 2) with 3) under the substitution Socrates | x yields:

4) ~Mortal (Socrates).



Note that we have assumed that both clauses 2) and 3) are true. If clause 3) is true, then the only reason clause 2) can be true is if ~Mortal(Socrates) is true. Finally, by combining 1) with 4) we derive (ϕ), in other words, the empty clause, and so we have a contradiction. Therefore, the negation of what was assumed true, in other words, not (~ Will _ Die (Socrates)), which is equivalent to Will_Die (Socrates), must be true. The original conclusion does logically follow from the argument's premises, and therefore the argument is valid.

Example-1

1. All great chefs are Italian.
2. All Italians enjoy good food.
3. Either Michael or Louis is a great chef.
4. Michael is not a great chef.
5. Therefore, Louis enjoys good food.

Express in resolution tree

We use the following predicates:

GC (x) : x is a great chef
I (x) : x is Italian
EF (x) : x enjoys good food

Using these symbols, we can express the given statements in the predicate logic as :

1. (∀x) (GC (x) ⇒ I (x))
2. (∀x) (I (x) ⇒ EF (x))
3. GC (Michael) ∨ GC (Louis)
4. ~ GC (Michael)

Therefore:

5. EF (Louis)

To use resolution, we first need to convert the premises (1-4) into conjunctive normal form (CNF).

1. ~ GC (x) ∨ I (x)
2. ~ I (x) ∨ EF (x)
3. GC (Michael) ∨ GC (Louis)
4. ~ GC (Michael)

Negate the conclusion:

5. ~ EF (Louis) // already in clause form

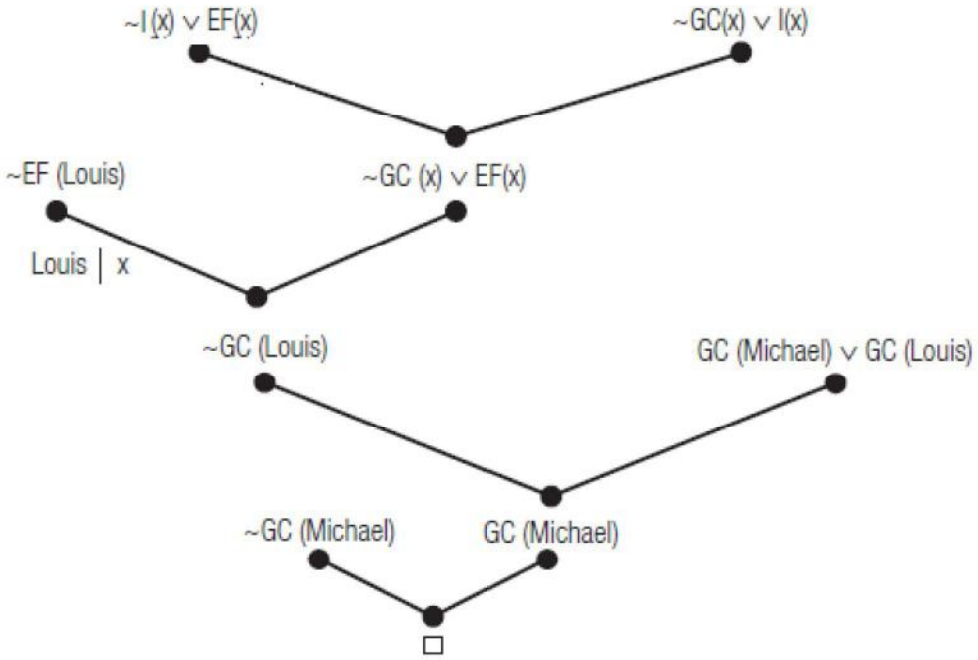


Figure: Resolution proof —a graphical representation.

We display the search for a contradiction in graphical form in Figure above. The substitutions made are shown on the branches.

Example-2:

Marcus was a man. Marcus was a Pompeian. All Pompeians are Romans. Caesar was a ruler. All Romans are either loyal to Caesar or hated him. Everyone is loyal to someone. People only try to kill rulers they are not loyal to. Marcus tried to kill Caesar.

Prove that Marcus hates Caesar. So we start with the negation that Marcus does not hate Caesar and then refute that claim by looking for conflict in the database

Convert into clausal form

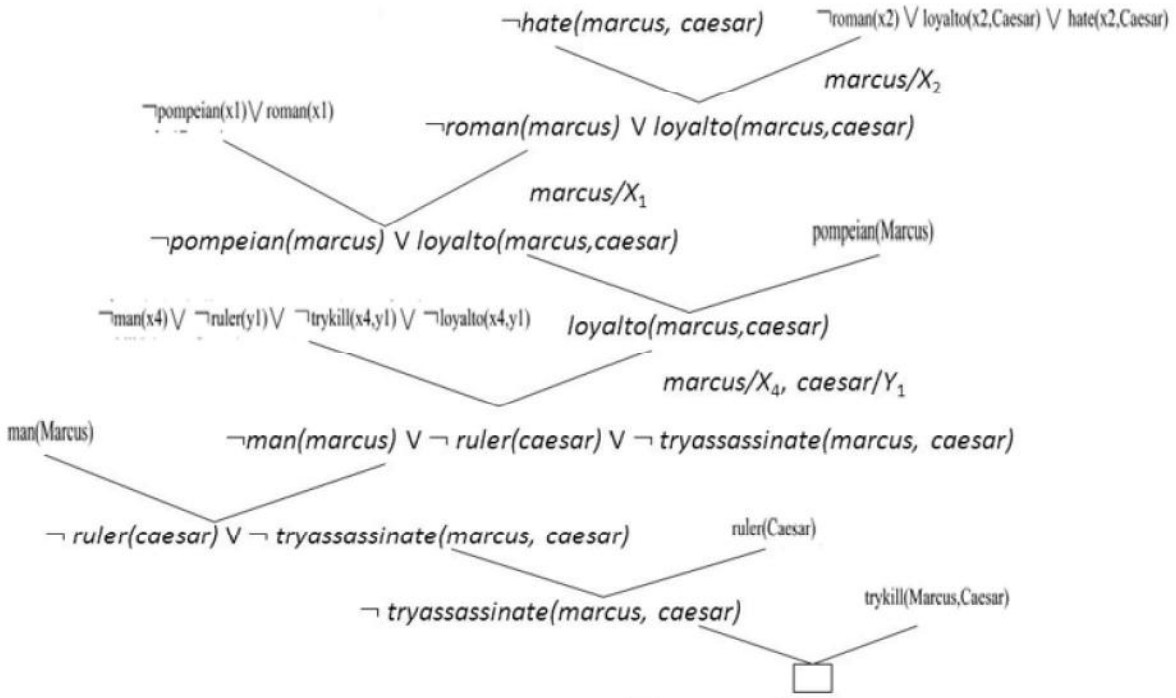
1 man(Marcus)
2 pompeian(Marcus)
3 $\forall$ x[pompeian(x) => roman(x)]
4 ruler(Caesar)
5 $\forall$ x[roman(x) => loyalto(x,Caesar) $\vee$ hate(x,Caesar)]
6 $\forall$ x $\exists$ yloyalto(x,y)
7 $\forall$ x $\forall$ y man(x) $\wedge$ ruler(y) $\wedge$ trykill(x,y) => $\neg$ loyalto(x,y)
8 trykill(Marcus,Caesar)

They are converted to conjunctive normal form as

1 man(Marcus)
2 pompeian(Marcus)
3 $\neg$pompeian(x1) $\vee$ roman(x1)
4 ruler(Caesar)
5 $\neg$roman(x2) $\vee$ loyalto(x2,Caesar) $\vee$ hate(x2,Caesar)
6 loyalto(x3,S1(x3)) *(Skolem func)*
7 $\neg$man(x4) $\vee$ $\neg$ruler(y1) $\vee$ $\neg$trykill(x4,y1) $\vee$ $\neg$loyalto(x4,y1)
8 trykill(Marcus,Caesar)

9 $\neg$hate(Marcus,Caesar)

Resolution Proof



Example-3

- Anyone passing his history exams and winning the lottery is happy
- Anyone who studies or is lucky can pass all his exams.
- John did not study but he is lucky.
- Anyone who is lucky wins the lottery.

Prove that

John is happy

⇨ $\forall$ X (pass (X,history) $\wedge$ win (X,lottery) $\rightarrow$ happy (X))

⇨ $\forall$ X $\forall$ Y (study (X) $\vee$ lucky (X) $\rightarrow$ pass (X,Y))

⇨ $\neg$ study (john) $\wedge$ lucky (john)

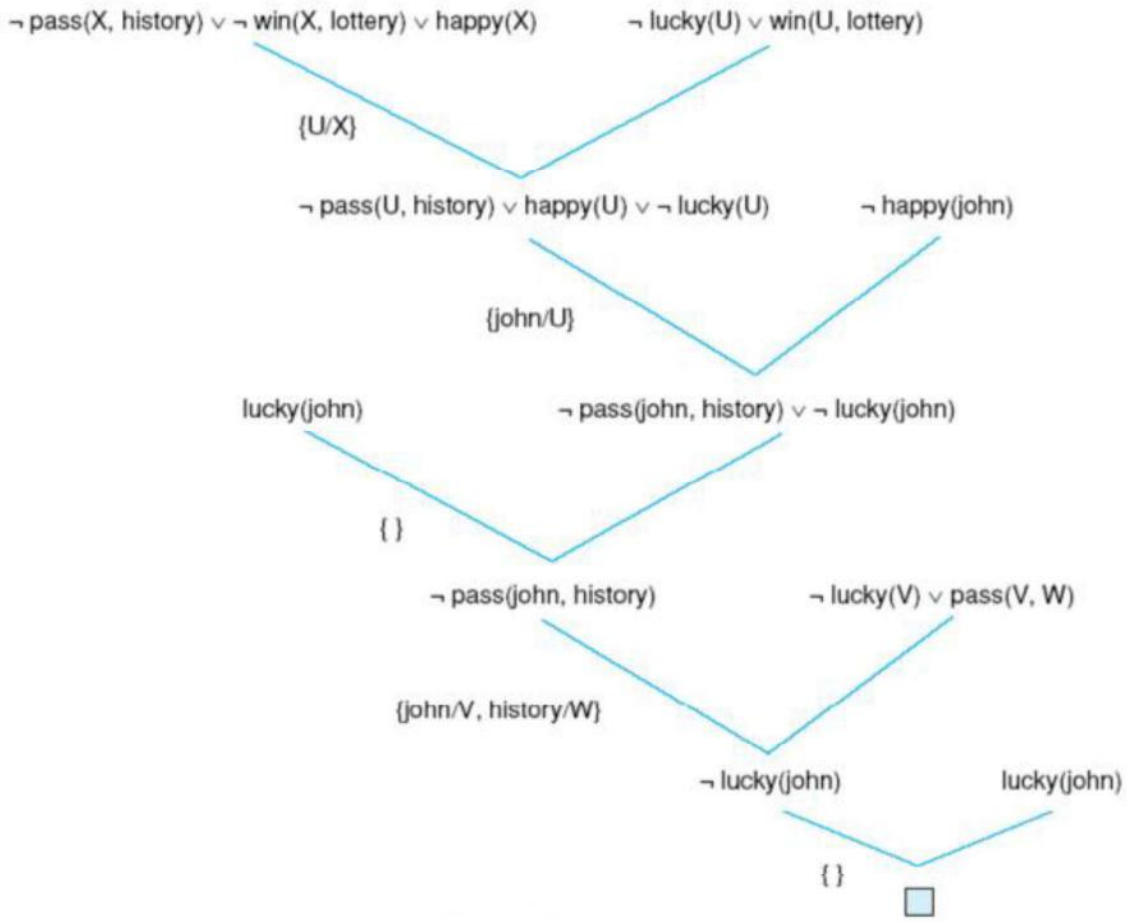
⇨ $\forall$ X (lucky (X) $\rightarrow$ win (X,lottery))

These four predicate statements are now changed to clause form

1. $\neg$ pass (X, history) $\vee$ $\neg$ win (X, lottery) $\vee$ happy (X)
2. $\neg$ study (Y) $\vee$ pass (Y, Z)
3. $\neg$ lucky (W) $\vee$ pass (W, V)
4. $\neg$ study (john)
5. lucky (john)
6. $\neg$ lucky (U) $\vee$ win (U, lottery)

Into these clauses is entered, in clause form, the negation of the conclusion:

7. $\neg$ happy (john)



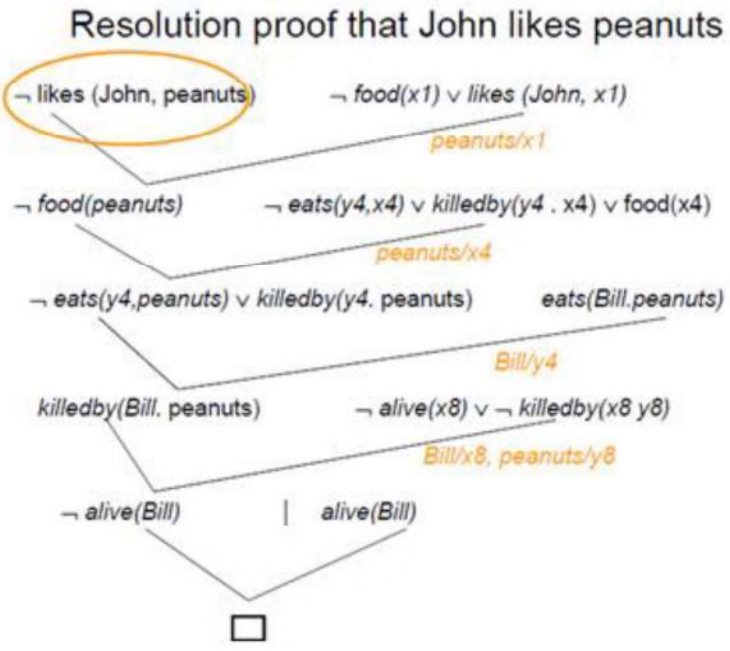
Example:-

1. **John likes all kinds of food.**
2. **Apples are food.**
3. **Chicken is food.**
4. **Anything anyone eats and isn't killed by is food.**
5. **Bill eats peanuts and is still alive.**
6. **Sue eats everything Bill eats.**
7. $\forall x:\forall y: alive(x) \rightarrow \neg killedby(x,y)$

proof that John likes peanuts

1. $\forall x$: food(x) $\rightarrow$ likes(john, x)
2. food(apple)
3. food(chicken)
4. $\forall x$:($\exists y$: eats(y, x) $\wedge \neg$ killedby(y, x)) $\rightarrow$ food(x)
5. A. eats(Bill, peanuts) B. alive(Bill)
6. $\forall x$:eats(Bill, x)$\rightarrow$ eats(Sue, x)
7. $\forall x$:$\forall y$: alive(x)$\rightarrow \neg$ killedby (x,y)

1. $\neg$ food(x1) $\vee$ likes(John, x1)
2. food(apples)
3. food(chicken)
4. $\neg$ eats(y4,x4) $\vee$ killedby(y4 , x4) $\vee$ food(x4)
5. Eats (Bill, peanuts)
6. Alive (Bill)
7. $\neg$ eats(Bill,x7) $\vee$ eats(Sue,x7)
8. $\neg$ alive(x8) $\vee \neg$ killedby(x8, y8)



Board Question:

2076

Convert following statement into FOPL,

1. Every friend of Ramesh has visited pokhara.
2. Everyone who visits Pokhara does boating on Fewa lake.
3. Ramesh has done boating on Fewa lake.

Now using resolution try to infer; some friend of Ramesh has done boating on Fewa lake

Solution:

Step1:

To infer using resolution, we first convert the statements into their logical form:

1. $\forall x$ (Friend(x, Ramesh) $\rightarrow$ Visited(x, Pokhara))
2. $\forall y$ (Visited(y, Pokhara) $\rightarrow$ Boating(y, Fewa))
3. Boating(Ramesh, Fewa)

Goal: $\exists z$ (Friend(z, Ramesh) $\wedge$ Boating(z, Fewa))

Step 2:

Convert the premises into conjunctive normal form (CNF).

1. $\neg$Friend(x, Ramesh) $\vee$ Visited(x, Pokhara),
2. $\neg$Visited(y, Pokhara) $\vee$ Boating(y, Fewa),
3. Boating(Ramesh, Fewa),

Step 3:

We then negate the conclusion we want to infer:
$\neg(\exists z$ (Friend(z, Ramesh) $\wedge$ Boating(z, Fewa)))

Which can be equivalent to
Friend(z, Ramesh),
$\neg$Boating(z, Fewa)

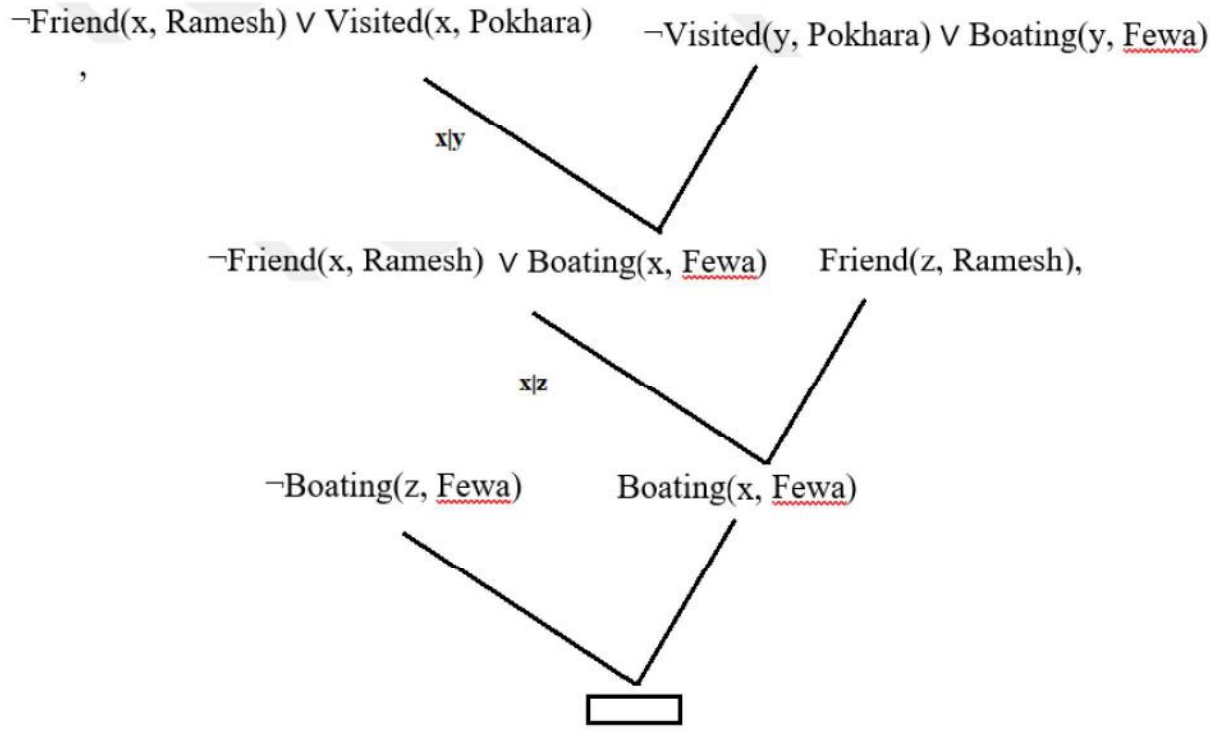
Now,

We then listing out all the premises and the negation:

- $\neg$Friend(x, Ramesh) $\vee$ Visited(x, Pokhara),
- $\neg$Visited(y, Pokhara) $\vee$ Boating(y, Fewa),
- Boating(Ramesh, Fewa),
- Friend(z, Ramesh),
- $\neg$Boating(z, Fewa)

We then perform resolution on all possible pairs of clauses until we either derive the empty clause (meaning that the original set of clauses is unsatisfiable, and thus the conclusion is true), or we cannot perform any further resolutions (meaning that we cannot derive the conclusion from the premises).

Here is the resolution tree:



Question: 2076 new

How resolution algorithm is used in FOPL to infer conclusion?
Consider the facts;

- Anyone whom pugu loves is a star.
- Any hero who does not rehearse does not act.
- Anmol is a hero.
- Any hero who does not work does not rehearse.
- Anyone who does not act is not a star.

Convert above statement into FOPL and use resolution to infer that "If Anmol does not work, then pugu does not love Anmol"

2075

Consider the following statements.

All cats like fish, cats eat everything they like, and Ziggy is a cat.

a) Translate the sentences into FOL.
b) Convert the sentences into clausal normal form.
c) Answer using FOL, if Ziggy eats fish?

2074

Convert following sentences to FOPL.

1. If every helper is busy then there is a job in the queue.
2. A job is in queue but the helper is not busy.
3. Every helpers are teased by someone

2073

8-Consider the following statements:

- Rabin likes only easy courses. Science courses are hard.
- All courses in the CSIT are easy.
- CSC 101 is a CSIT course.

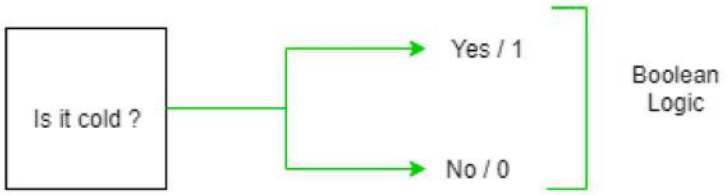
a. Translate the sentences into predicate logic.
b. Convert your sentences into clausal normal form (CNF).

Fuzzy Logics

- The term fuzzy logic was introduced in 1965 with the proposal of fuzzy set theory by Lotfi Zadeh. The term fuzzy refers to things which are not clear or are vague.

- Fuzzy logic is a form of many-valued logic in which the truth values of variables may be any real number between 0 and 1 both inclusive. It is employed to handle the concept of partial truth, where the truth value may range between completely true and completely false. By contrast, in Boolean logic, the truth values of variables may only be the integer values 0 or 1.

- In Boolean system truth value, 1.0 represents absolute truth value and 0.0 represents absolute false value. But in the fuzzy system, there is no logic for absolute truth and absolute false value. But in fuzzy logic, there is intermediate value to present which is partially true and partially false.



- Fuzzy logic is based on the observation that people make decisions based on imprecise and non-numerical information. Fuzzy models or sets are mathematical means of representing vagueness and imprecise information (hence the term fuzzy). These models have the capability of recognising, representing, manipulating, interpreting, and utilising data and information that are vague and lack certainty.

Example

- The design of a fuzzy logic system starts with a set of **membership functions** for each input and a set for each output. A membership function is simply a graphical representation of the fuzzy variable sets. A set of rules is then applied to the membership functions to yield a "crisp" output value.

- Consider TEMPERATURE is the input and FAN SPEED is the output. To create a set of membership functions for each input, it uses three fuzzy sets, COLD, WARM, and HOT. Then We will create a membership function for each of three sets of temperature as shown in the cold-normal-hot graphic, Figure 1.

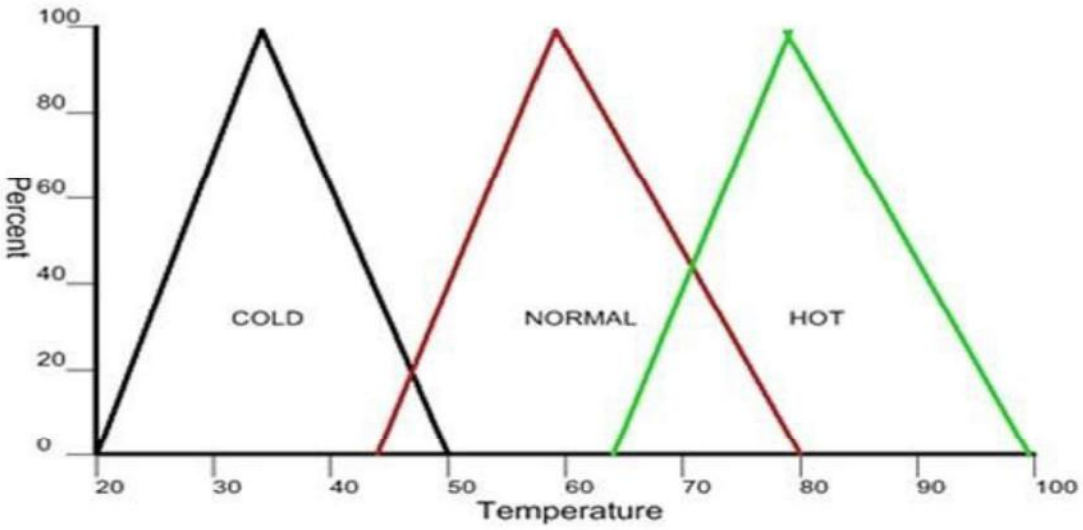


Figure 1

- We will use three fuzzy sets for the output, SLOW, MEDIUM, and FAST. A set of functions is created for each output set just as for the input sets.

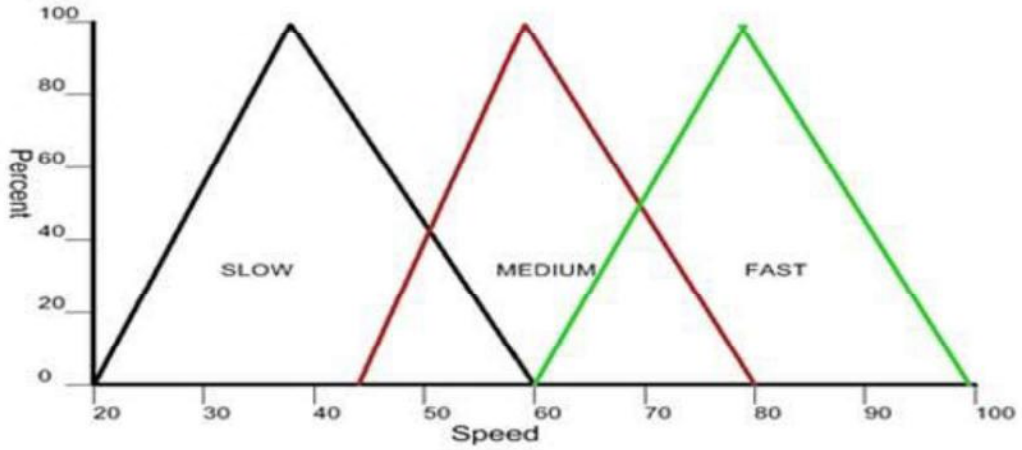


Figure 2

Now that we have our membership functions defined, we can create the rules that will define how the membership functions will be applied to the final system. We will create three rules for this system.

- If HOT then FAST
- If WARM then MEDIUM
- If COLD then SLOW

The rules are then applied to the membership functions to produce the "crisp" output value to drive the system. For simplicity we will illustrate using only two input and two output functions. For an input value of 52 degrees, we intersect the membership functions. We see that in this example the intersection will be on both functions, thus two rules are applied. The intersection points are extended to the output functions to produce an intersecting point. The output functions are then truncated at the height of the intersecting points. The area under the curves for each membership function is then added to give us a total area. The centroid of this area is calculated. The output value is then the centroid value. In this example 44% is the output FAN SPEED value.

Fuzzy logic has been applied to many fields as explained below

- It is used in the aerospace field for altitude control of spacecraft and satellite.
- It has used in the automotive system for speed control, traffic control.
- It is used for decision making support systems and personal evaluation in the large company business.
- It has application in chemical industry for controlling the pH, drying, chemical distillation process.
- Fuzzy logic is used in Natural language processing and various intensive applications in Artificial Intelligence.
- Fuzzy logic is extensively used in modern control systems such as expert systems.
- Fuzzy Logic is used with Neural Networks as it mimics how a person would make decisions, only much faster. It is done by Aggregation of data and changing into more meaningful data by forming partial truths as Fuzzy sets.

Handling Uncertain Knowledge

In real life, it is not always possible to determine the state of the environment as it might not be clear. Agents may need to handle uncertainty, whether due to partial observability, nondeterminism, or a combination of the two. An agent may never know for certain what state it's in or where it will end up after a sequence of actions.

We have seen problem-solving agents and logical agents designed to handle uncertainty by keeping track of a **belief state** ("representation of the set of all possible world states that it might be in") and generating a contingency plan that handles every possible eventuality that its sensors may report during execution.

Despite its many virtues, however, this approach has significant drawbacks when taken literally as a recipe for creating agent programs:

- When interpreting partial sensor information, a logical agent must consider every logically possible explanation for the observations, no matter how unlikely. This leads to impossible large and complex belief-state representations.
- A correct contingent plan that handles every eventuality can grow arbitrarily large and must consider arbitrarily unlikely contingencies.
- Sometimes there is no plan that is guaranteed to achieve the goal—yet the agent must act. It must have some way to compare the merits of plans that are not guaranteed.

Suppose, for example, that an automated taxi has the goal of delivering a passenger to the airport on time. The agent forms a plan, A90, that involves leaving home 90 minutes before the flight departs and driving at a reasonable speed. Even though the airport is only about 5 miles away, a logical taxi agent will not be able to conclude with certainty that "Plan A90 will get us to the airport in time." Instead, it reaches the weaker conclusion "Plan A90 will get us to the airport in time, as long as the car doesn't break down or run out of gas, whether it doesn't get into an accident, and there are no accidents on the bridge, and the plane doesn't leave early, and no meteorite hits the car, and ....". So, none of these conditions can be deduced for sure, so the plan's success cannot be inferred. This is the qualification problem for which we have seen does not contain real solution.

Nonetheless, in some sense A90's agent's performance measure can be maximized. The performance measure includes getting to the airport in time for the flight, avoiding a long, unproductive wait at the airport, and avoiding speeding tickets along the way. The agent's knowledge cannot guarantee any of these outcomes for A90, but it can provide some degree of belief that they will be achieved. Other plans, such as A180, might increase the agent's belief that it will get to the airport on time, but also increase the likelihood of a long wait. **The right thing to do is to take rational decision…**

Rational decisions

Consider again the A90 plan for getting to the airport. Suppose it gives us a 97% chance of catching our flight. Does this mean it is a rational choice? Not necessarily: there might be other plans, such as A180, with higher probabilities. If it is vital not to miss the flight, then it is worth risking the longer wait at the airport. What about A1440, a plan that involves leaving home 24

hours in advance? In most circumstances, this is not a good choice, because although it almost guarantees getting there on time, it involves an intolerable wait—not to mention a possibly unpleasant diet of airport food.

To make such choices, an agent must first have preferences between the different possible outcomes of the various plans. An outcome is a completely specified state, including such factors as whether the agent arrives on time and the length of the wait at the airport.

We use **utility theory** to represent and reason with preferences. Utility theory says that every state has a degree of usefulness, so the agent will prefer states with higher utility. Preferences, as expressed by utilities, are combined with probabilities in the general theory of **rational decisions** called decision theory:

Decision theory = probability theory + utility theory.

The fundamental idea of decision theory is that an agent is rational if and only if it chooses the action that yields the highest expected utility, averaged over all the possible outcomes of the action. This is called the principle of maximum expected utility (MEU).

Example:

Let's consider the following simple rule for uncertain reasoning: diagnosing a dental patient's toothache.

$$\text{Toothache} \Rightarrow \text{Cavity}.$$

The problem is that this rule is wrong. Not all patients with toothaches have cavities; some of them have gum disease, an abscess, or one of several other problems:

$$\text{Toothache} \Rightarrow \text{Cavity} \vee \text{GumProblem} \vee \text{Abscess} \ldots$$

Unfortunately, in order to make the rule true, we have to add an almost unlimited list of possible problems. We could try turning the rule into a causal rule:

$$\text{Cavity} \Rightarrow \text{Toothache}$$

But this rule is not right either; not all cavities cause pain. The only way to fix the rule is to make it logically exhaustive: to augment the left-hand side with all the qualifications required for a cavity to cause a toothache. Trying to use logic to cope with a domain like medical diagnosis thus fails for three main reasons:

- **Laziness**: It is too much work to list the complete set of antecedents or consequents needed to ensure an exceptionless rule and too hard to use such rules.
- **Theoretical ignorance:** Medical science has no complete theory for the domain.
- **Practical ignorance:** Even if we know all the rules, we might be uncertain about a particular patient because not all the necessary tests have been or can be run.

The connection between toothaches and cavities is just not a logical consequence in either direction. This is typical of the medical domain, as well as most other judgmental domains. **The agent's knowledge can at best provide only a degree of belief in the relevant sentences**. Our main tool for dealing with degrees of belief is **probability theory**.

Probability provides a way of summarizing the uncertainty that comes from our laziness and ignorance, thereby solving the qualification problem. We might not know for sure what afflicts a particular patient, but we believe that there is, say, an 80% chance that is, a probability of 0.8% that the patient who has a toothache has a cavity.

One confusing point is that at the time of our diagnosis, there is no uncertainty in the actual world: the patient either has a cavity or doesn't. So, what does it mean to say the probability of a cavity is 0.8? Shouldn't it be either 0 or 1? The answer is that probability statements are made with respect to a knowledge state, not with respect to the real world.

We say "The probability that the patient has a cavity, given that she has a toothache, is 0.8." If we later learn that the patient has a history of gum disease, we can make a different statement: "The probability that the patient has a cavity, given that she has a toothache and a history of gum disease, is 0.4." If we gather further conclusive evidence against a cavity, we can say "The probability that the patient has a cavity, given all we now know, is almost 0." Note that these statements do not contradict each other; each is a separate assertion about a different knowledge state.

Hence in conclusion:

An agent working in real environment almost never has to whole truth about its environment. Therefore, the agent needs to work under uncertainty.

With knowledge representation, we might A-> B, which means if A is true then B is true, but consider a situation where we are not sure about whether A is true or not then in such a situation we cannot express this statement, this situation is called uncertainty.

When agent works with uncertain knowledge then it might be impossible to construct a complete and correct description of how its actions will work.

So, uncertainty is a common phenomenon in everyday interaction and in all areas of human lives especially when dealing with information from different sources. This information may be unreliable based on their sources or method of collection such as random sampling or other statistical methods rather than categorical means.

Uncertainty may arise from incomplete data or information, ambiguous and inconsistent data or information. In most tasks that requires intelligent behaviour, the problem of uncertainty cannot be completely ruled out. For example, tasks such as planning, reasoning, complex problem solving, decision-making and classification problems there are elements of uncertainty to some

extent because all the tasks require intelligence; may it be humans or machines. Even for machine which are expert systems, the software is developed by human experts and they are also liable to errors.

So, in summary…

- Uncertainty describes a situation involving ambiguous and/or unknown information.
- In real world problems, due to partially/non-partially observable, non-deterministic environment, or even due to the combination of the two, the agents may need to handle uncertainty.
- An agent may never know for certain what state it's in or where it will end up after a sequence of actions.
- The lack of knowledge or insufficient information that causes uncertainty can be due to: Vagueness, Conflict in information, No specificity.
- Thus, uncertainty can be described as a situation where information available to decision makers is imprecise to be summarized by a probabilistic nature.

The different causes of uncertainty are…

- The environment is not fully observable.
- The environment behaves in a non-deterministic way.
- The actions performed may not have desired effect.
- There might be unjustified reliance on assumptions.

The different sources of uncertainty are…

- **Uncertain Data:**
  - It includes missing, noisy, inconsistent, ambiguous, unreliable data.
- **Uncertain Knowledge:**
  - It contains an incomplete knowledge of the domain.
  - Multiple causes lead to multiple effects
  - Theoretical knowledge
  - Practical ignorance
- **Uncertain Knowledge Representation:**
  - It is nothing but representation providing restrictive model of the real system, data with imprecise representation & limited expressiveness in representation mechanism.
- **Inference process:**
  - The derived inference might be formally correct but wrong in the real world. The new conclusions may not be well founded.

Prior and Posterior Probability

Probability

Probability defines the likelihood of occurrence of an event. There are many real-life situations in which we may have to predict the outcome of an event. We may be sure or not sure of the results of an event. In such cases, we say that there is a probability of this event to occur or not occur. Probability generally has great applications in games, in business to make probability-based predictions, and also **probability has extensive applications in new area of artificial intelligence.**

Probability can be defined as the ratio of the number of favourable outcomes to the total number of outcomes of an event. For an experiment having 'n' number of outcomes, the number of favourable outcomes can be denoted by x.
The formula to calculate the probability of an event is as follows.

Probability (Event) = Favourable Outcomes/Total Outcomes = x/n

Terminology of Probability Theory

Experiment: A trial or an operation conducted to produce an outcome is called an experiment.

Sample Space: All the possible outcomes of an experiment together constitute a sample space. For example, the sample space of tossing a coin is head and tail. The sample space is denoted by **S** or Greek letter omega (Ω). The number of elements in S is denoted by n(S). A possible outcome is also called a sample point since it is an element in the sample space

Favourable Outcome: An event that has produced the desired result or expected event is called a favourable outcome. For example, when we roll two dice, the possible/favourable outcomes of getting the sum of numbers on the two dice as 4 are (1,3), (2,2), and (3,1).

Trial: A trial denotes doing a random experiment.

Random Experiment: An experiment that has a well-defined set of outcomes is called a random experiment. For example, when we toss a coin, we know that we would get ahead or tail, but we are not sure which one will appear.

Event: The total number of outcomes of a random experiment is called an event.

Equally Likely Events: Events that have the same chances or probability of occurring are called equally likely events. The outcome of one event is independent of the other. For example, when we toss a coin, there are equal chances of getting a head or a tail.

Exhaustive Events: When the set of all outcomes of an experiment is equal to the sample space, we call it an exhaustive event.

Mutually Exclusive Events: Events that cannot happen simultaneously are called mutually exclusive events. For example, the climate can be either hot or cold. We cannot experience the same weather simultaneously.

Properties of Probability

- For an event A, its probability is defined as P(A). The probability of an event A is calculated by the following formula:

$$P(A) = \frac{Number\ \ of\ \ favorable\ \ outcomes}{Total\ \ number\ \ of\ \ outcomes}$$

- If there is no ambiguity in the occurrence of an event, then the probability of such an event is equal to 1. In other words, the probability of a certain event is 1.
- If an event has no chances of occurring, then its probability is 0.
- The probability of an event A is depicted by a number P (A) in such a way that

$$0 \leq P(A) \leq 1$$

In short, the probability is always a positive number.

- For two mutually exclusive events R and S,

P(R U S) = P(R) + P(S).

- The event that has only one outcome is known as an elementary event. The sum of probabilities of elementary events is equal to 1.
  - P (R U S) = P(R) + P(S) - P (R ∩ S)
  - P (R ∩ S) = P(R) + P(S) - P(RUS)

- For mutually exclusive events $R_1, R_2, R_3, ..., R_n$

, $P(R_1 U R_2 U R_3, ...U R_n)$ is equal to $P(R_1) + P(R_2), +.....P(R_n)$

Prior Probability

Prior is a probability calculated to express one's beliefs about this quantity before some evidence is taken into account. The prior probability is the probability assigned to an event before the arrival of some information that makes it necessary to revise the assigned probability.

A priori probability, also known as classical probability, is deduced from formal reasoning. In other words, a priori probability is derived from logically examining an event.
Formula for A Priori Probability

$$\text{A Prior Probability} = F/N$$

Where:

- f refers to the number of desirable outcomes.
- N refers to the total number of outcomes.

Note that the formula above can only be used for events where outcomes all have equal odds of occurring and are mutually exclusive.

Formal Reasoning in A Priori Probability

A priori probability requires formal reasoning. For example, consider a coin toss. What is the a priori probability of a head in a single coin toss?

One can argue that given a coin has two sides, both of which have equal surface areas, that it is symmetrical. Ignoring the possibility of a coin landing on its edge and staying there, it would suggest that the probability of a coin landing on heads is the same as a coin landing on tails. Therefore, the a priori probability of a coin toss landing on heads is equal to a coin toss landing on tails, which is 50%.

Examples of A Priori Probability

Example 1: Fair Dice Roll

A six-sided fair dice is rolled. What is the a priori probability of rolling a 2, 4, or 6, in a dice roll?

The number of desired outcomes is 3 (rolling a 2, 4, or 6), and there are 6 outcomes in total. The a priori probability for this example is calculated as follows:

A priori probability = 3 / 6 = 50%.

Therefore, the a priori probability of rolling a 2, 4, or 6 is 50%.

Example 2: Deck of Cards

In a standard deck of cards, what is the a priori probability of drawing an ace of spades?

The number of desired outcomes is 1 (an ace of spades), and there are 52 outcomes in total. The a priori probability for this example is calculated as follows:

A priori probability = 1 / 52 = 1.92%.

Therefore, the a priori probability of drawing the ace of spades is 1.92%.

Posterior Probability

A posterior probability is the updated probability of some event occurring after accounting for new information.

For example, we might be interested in finding the probability of some event "A" occurring after we account for some event "B" that has just occurred. We could calculate this posterior probability by using the following formula:

$$P(A|B) = P(A) * P(B|A) / P(B)$$

where:

- P(A|B) = the probability of event A occurring, given that event B has occurred. Note that "|" means "given."
- P(A) = the probability that event A occurs.
- P(B) = the probability that event B occurs.
- P(B|A) = the probability of event B occurring, given that event A has occurred.

Conditional Probability

Conditional probability is known as the possibility of an event or outcome happening, based on the existence of a previous event or outcome. It is calculated by multiplying the probability of the preceding event by the renewed probability of the succeeding, or conditional, event.

Conditional Probability Formula

$P(B|A) = P(A \text{ and } B) / P(A)$

Or:

$P(B|A) = P(A \cap B) / P(A)$

Where

P = Probability

A = Event A

B = Event B

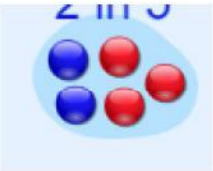
Here the concept of the independent event and dependent event occurs.

Independent Events
Events can be "Independent", meaning each event is not affected by any other events.

Example: Tossing a coin.
Each toss of a coin is a perfect isolated thing.
What it did in the past will not affect the current toss.
The chance is simply 1-in-2, or 50%, just like ANY toss of the coin.
So, each toss is an Independent Event.

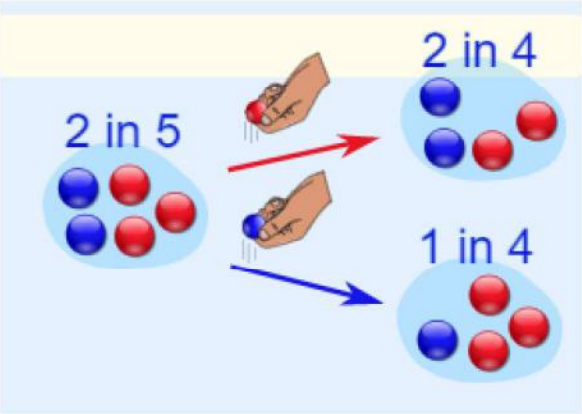
Dependent Events
But events can also be "dependent" ... which means they can be affected by previous events ...

Example: Marbles in a Bag

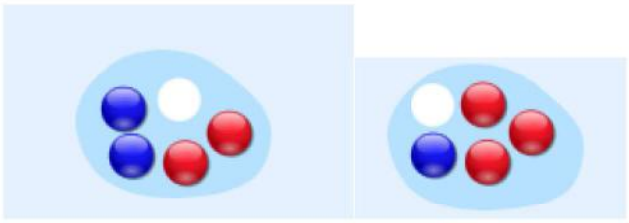


- 2 blue and 3 red marbles are in a bag.
- What are the chances of getting a blue marble?
  - ✓ The chance is 2 in 5

But after taking one out the chances change!



So the next time:



- if we got a red marble before, then the chance of a blue marble next is 2 in 4
- if we got a blue marble before, then the chance of a blue marble next is 1 in 4
- This is because we are removing marbles from the bag.
- So, the next event depends on what happened in the previous event, and is called dependent.

Note: if we replace the marbles in the bag each time, then the chances do not change and the events are independent:

- With Replacement: the events are Independent (the chances don't change)
- Without Replacement: the events are Dependent (the chances change)

Ex : Find the probability that a single toss of a dice will result in a number less than 4 if it is given that the toss resulted in an odd number.

Solution:

Let event A: toss resulted in an odd number and

Event B: number is less than 4

$\therefore A = \{1, 3, 5\}$

$\therefore P(A) = 3/6 = 1/2$

$B = \{1,2,3\}$

$\therefore A \cap B = \{1,3\}$

$P(A \cap B) = 2/6 = 1/3$

$\therefore$ P (number is less than 4 given that it is odd)

$= P(B/A) = P(A \cap B)/P(A) = (1/3) / (1/2) = 2/3$

Radom Variables

- A random variable is a variable whose value is unknown or a function that assigns values to each of an experiment's outcomes.
- Variables in probability theory are called random variables and their names begin with an uppercase letter.
- A random variable can be either discrete (having specific values) or continuous (any value in a continuous range).
- Every random variable has a domain—the set of possible values it can take on.
- The domain of Total for two dice is the set {2,..., 12} and the domain of Die1 is {1,..., 6}. A Boolean random variable has the domain {true, false}
- The use of random variables is most common in probability and statistics, where they are used to quantify outcomes of random occurrences.

- Risk analysts use random variables to estimate the probability of an adverse event occurring.

By convention, propositions of the form A = true are abbreviated simply as a, while A = false is abbreviated as ¬a. We can combine sorts of elementary propositions by using the connectives of propositional logic.

Example:
"The probability that the patient has a cavity, given that she is a teenager with no toothache, is 0.1" as follows:

P(cavity | ¬toothache ∧ teen)=0.1 .

Sometimes we will want to talk about the probabilities of all the possible values of a random variable. We could write:
P(Weather = sunny)=0.6
P(Weather = rain)=0.1
P(Weather = cloudy)=0.29
P(Weather = snow)=0.01 ,

but as an abbreviation we will allow

P(Weather) = (0.6, 0.1, 0.29, 0.01),

Where the bold P indicates that the result is a vector of numbers, and where we assume a predefined ordering (sunny, rain, cloudy, snow) on the domain of Weather.

We say that the P statement defines a probability distribution for the random variable Weather. The P notation is also used for conditional distributions: P(X | Y) gives the values of

P(X = xi | Y = yj) for each possible i, j pair.

Joint probability Distribution

Joint probability is a statistical measure that concludes the likelihood of two events occurring together at the same point in time. Joint probability is the probability of event Y occurring at the same time that event X occurs.

To represent the joint probability distribution the following formula can be used.

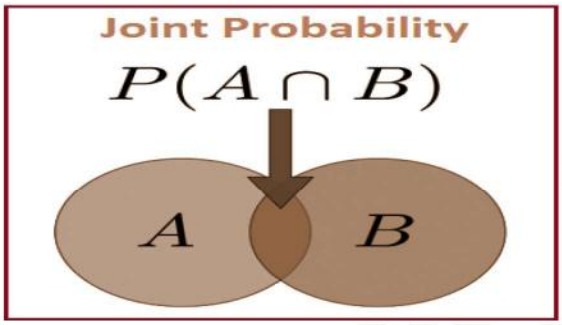
P (A∩B)

where,

- A, B= Two events

- P(A and B) or
- P(A,B)=The joint probability of A and B

The symbol "∩" in a joint probability is called an intersection. The probability of event A and event B happening is the same thing as the point where A and B intersect. Hence, the joint probability is also called the intersection of two or more events. We can represent this relation using a Venn diagram as shown below.



Example 1.

Find the probability that the number three will occur twice when two dice are rolled at the same time.

Solution:

Number of possible outcomes when a die is rolled = 6
i.e. {1, 2, 3, 4, 5, 6}
Let A be the event of occurring 3 on first die and B be the event of occurring 3 on the second die.
Both the dice have six possible outcomes, the probability of a three occurring on each die is 1/6.
P(A) =1/6
P(B)=1/6
P(A,B) = 1/6 x 1/6 = 1/36

Joint Probability Table

A joint probability distribution represents a probability distribution for two or more random variables. Instead of events being labelled A and B, the condition is to use X and Y as given below.
F(x,y) = P(X = x, Y = y)

The main purpose of this is to look for a relationship between two variables. For example, the below table shows some probabilities for events X and Y happening at the same time:

Now, Roll two dice.

Let X be the value on the first die and let Y be the value on the second die. Then both X and Y take values 1 to 6 and the joint pmf is p(i, j) = 1/6 * 1/6 = 1/36 for all i and j between 1 and 6. Here is the joint probability table:

$X\backslash Y$	1	2	3	4	5	6
1	1/36	1/36	1/36	1/36	1/36	1/36
2	1/36	1/36	1/36	1/36	1/36	1/36
3	1/36	1/36	1/36	1/36	1/36	1/36
4	1/36	1/36	1/36	1/36	1/36	1/36
5	1/36	1/36	1/36	1/36	1/36	1/36
6	1/36	1/36	1/36	1/36	1/36	1/36

Inference using Full Joint Distribution

We use the full joint distribution as the knowledge base from which answer to all question may be derived. The probability of a proposition is equal to the sum of the probabilities of the atomic events in which it holds.

$P(a)=\sum P(a_i)$

Therefore, given a full joint distribution that specifies the probabilities of all the atomic events, one can compute the probability of any proposition.

Probability of all possible worlds can be described using a table called a full joint probability distribution – the elements are indexed by values of random variables.

	toothache		¬ *toothache*	
	catch	¬ *catch*	*catch*	¬ *catch*
cavity	**.108**	**.012**	**.072**	**.008**
¬ *cavity*	**.016**	**.064**	**.144**	**.576**

Given the table, we can calculate probabilities of values of any random variable:
P(toothache=true) = 0.108 + 0.012 + 0.016 + 0.064 = 0.2
P(toothache=false) = 0.072+ 0.008 + 0.144 + 0.576 = 0.8

We will describe the table in a short way as: P(Toothache) = ⟨0.2, 0.8⟩

Example:

	toothache		¬ *toothache*	
	catch	¬ *catch*	*catch*	¬ *catch*
cavity	.108	.012	.072	.008
¬ *cavity*	.016	.064	.144	.576

$$P(\phi) = \Sigma_{\omega:\omega|=\phi} P(\omega)$$
$$P(Y) = \Sigma_{z \in Z} P(Y,z)$$

P(toothache) (= P(Toothache=true)) = 0.108 + 0.012 + 0.016 + 0.064 = 0.2
P(cavity ∨ toothache) = 0.108+0.012+0.072+0.008+0.016+0.064 = 0.28

P(¬cavity|toothache) = P(¬cavity ∧ toothache) / P(toothache)
= (0.016 + 0.064) / (0.108 + 0.012 + 0.016 + 0.064)
= 0.4

P(¬cavity|toothache) = P(¬cavity ∧ toothache) / P(toothache)
= (0.016 + 0.064) / (0.108 + 0.012 + 0.016 + 0.064)
= 0.4

P(cavity|toothache) = P(cavity ∧ toothache) / P(toothache)
= (0.108 + 0.012) / (0.108 + 0.012 + 0.016 + 0.064)
= 0.6

Bayes' Rule and its use

Baye's theorem is a mathematical formula for determining conditional probability or it is a way to apply conditional probability for prediction. Conditional probability is the likelihood of an outcome occurring, based on a previous outcome having occurred in similar circumstances. Mathematically, Baye's theorem can be expressed as…

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A) \cdot P(B|A)}{P(B)}$$

where:

P(A)= The probability of A occurring
P(B)= The probability of B occurring
P(A|B)=The probability of A given B
P(B|A)= The probability of B given A
P(A∩B))= The probability of both A and B occurring

Proof:

We know that,
P(a/b)= p(a∩b)/p(b)
p(a∩b)= P(a/b) *p(b) ……(i)

P(b/a)=p(b∩a)/p(a)
p(b∩a) =P(b/a) * p(a) ……. (ii)
Simillarly,

from equation ii and ii
P(a/b) *p(b) = P(b/a) * p(a) …… [p(b∩a) ≡ p(b∩a)]
P(a/b) = P(b/a) * p(a) / p(b)

Hence, Bayes theorem provides a way to revise existing predictions or theories (updated probabilities) given new additional evidence. This , in turn, make the predictions more accurate.

Bayesian Networks

A Bayesian network (BN) is a probabilistic graphical model for representing knowledge about an uncertain domain where each node corresponds to a random variable and each edge represents the conditional probability for the corresponding random variables. BNs are also called belief networks or Bayes nets. Due to dependencies and conditional probabilities, a BN corresponds to a directed acyclic graph (DAG) where no loop or self-connection is allowed.

For example:-

P(x)=0.5
P(y/x) =0.7
P(z)=0.8
P(u/y) =0.47

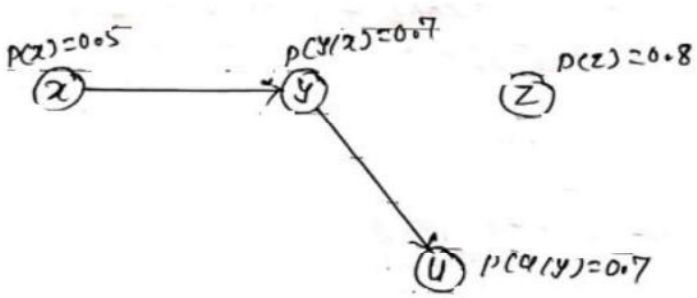


Fig: Bayesian Graph

Inference with Belief / Bayesian Network:

- Using a Bayesian network to compute probabilities is called inference in Bayesian network.
- The first task is to compute the posterior probability distribution for the query variable X, given some assignment of values e to the set of evidence variable E = E1,....., En and the hidden variables are Y = Y1,.....Yn.
- From the full joint probability distribution we can answer the query P(X/e) by computing

$$P(x/e) = \alpha P(X, e) = \alpha \Sigma Y(X, e, Y)$$

- A Bayesian network gives a complete representation of the full joint distribution, specifically, the terms P (X, e, Y) can be written as products of conditional probabilities from the network.
- Therefore, a query can be answered using a Bayesian network by computing sums of products of conditional probabilities from the network.

Example:

You have installed a new burglar alarm at your home to detect burglary. The alarm reliably responds at detecting a burglary but also responds for minor earthquakes. You have two neighbors John and marry, who have taken a responsibility to inform you at work when they hear the alarm. john always calls you when he hears the alarm, but sometimes he got confused with the phone ringing and calls at that time too. On the other hand, marry likes to listen to high music, so

sometimes she misses to hear the alarm. Here we would like to compute the probability of Burglary Alarm.

Problem:

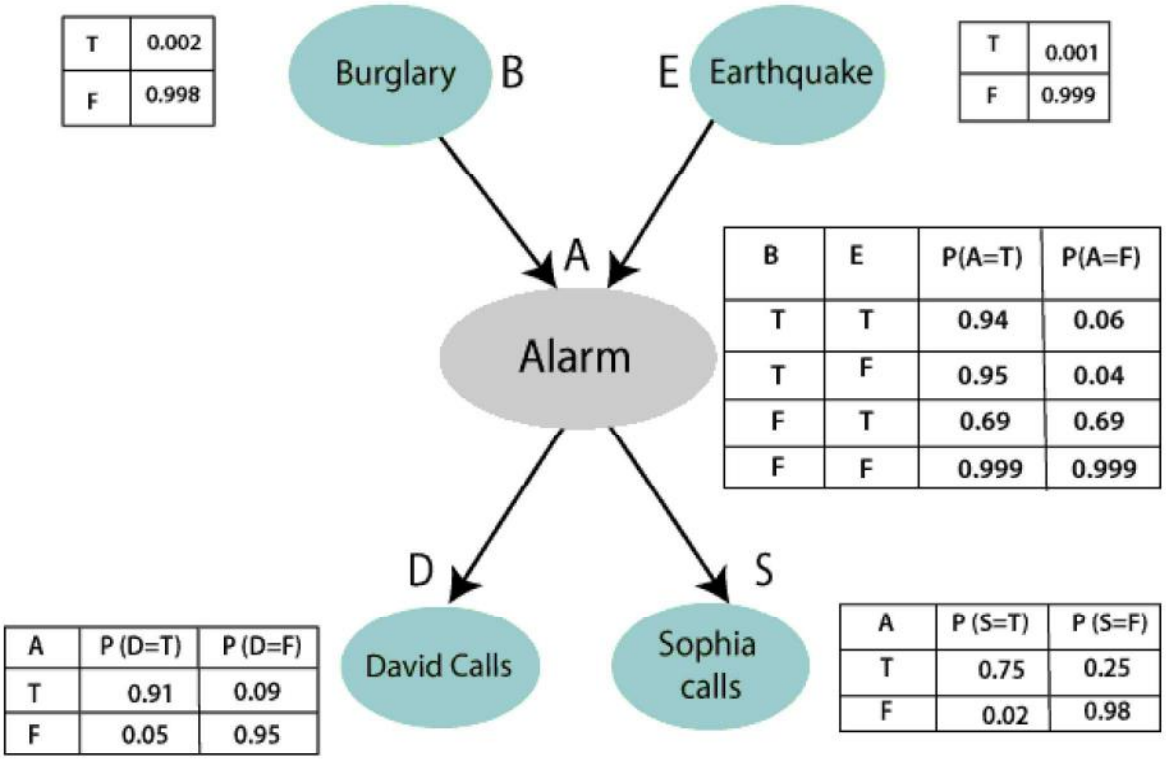
Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and john and marry both called you.

Solution:

- The Bayesian network for the above problem is given below. The network structure is showing that burglary and earthquake is the parent node of the alarm and directly affecting the probability of alarm's going off, but john and marry's calls depend on alarm probability.
- The network is representing that our assumptions do not directly perceive the burglary and also do not notice the minor earthquake, and they also not confer before calling.
- The conditional distributions for each node are given as conditional probabilities table(CPT).
- Each row in the CPT must be sum to 1 because all the entries in the table represent an exhaustive set of cases for the variable.
- In CPT, a boolean variable with k boolean parents contains 2K probabilities. Hence, if there are two parents, then CPT will contain 4 probability values

List of all events occurring in this network:

- Burglary (B)
- Earthquake(E)
- Alarm(A)
- David Calls(D)
- Sophia calls(S)



T	0.002
F	0.998

T	0.001
F	0.999

B	E	P(A=T)	P(A=F)
T	T	0.94	0.06
T	F	0.95	0.04
F	T	0.69	0.69
F	F	0.999	0.999

A	P (D=T)	P (D=F)
T	0.91	0.09
F	0.05	0.95

A	P (S=T)	P (S=F)
T	0.75	0.25
F	0.02	0.98

Conditional probability table for Alarm A:

The Conditional probability of Alarm A depends on Burglar and earthquake:

B	**E**	**P(A= True)**	**P(A= False)**
True	True	0.94	0.06
True	False	0.95	0.04
False	True	0.31	0.69
False	False	0.001	0.999

Conditional probability table for john Calls:

The Conditional probability of john that he will call depends on the probability of Alarm.

A	P(D= True)	P(D= False)
True	0.91	0.09
False	0.05	0.95

Conditional probability table for marry Calls:

The Conditional probability of marry that she calls is depending on its Parent Node "Alarm."

A	P(S= True)	P(S= False)
Truc	0.75	0.25
False	0.02	0.98

From the formula of joint distribution, we can write the problem statement in the form of probability distribution:

P(S, D, A, ¬B, ¬E)
= P (S|A) *P (D|A)*P (A|¬B ^ ¬E) *P (¬B) *P (¬E).
= 0.75* 0.91* 0.001* 0.998*0.999
= 0.00068045.

Hence, a Bayesian network can answer any query about the domain by using Joint distribution.

========== End Unit 4==========

Unit IV: Knowledge Representation (14 Hrs.)

4.1. Definition and importance of Knowledge, Issues in Knowledge Representation, Knowledge Representation Systems, Properties of Knowledge Representation Systems.

4.2. Types of Knowledge Representation Systems: Semantic Nets, Frames, Conceptual Dependencies, Scripts, Rule Based Systems, Propositional Logic, Predicate Logic

4.3. Propositional Logic(PL): Syntax, Semantics, Formal logic-connectives, truth tables, tautology, validity, well-formed-formula, Inference using Resolution, Backward Chaining and Forward Chaining

4.4. Predicate Logic: FOPL, Syntax, Semantics, Quantification, Inference with FOPL: By converting into PL (Existential and universal instantiation), Unification and lifting, Inference using resolution

4.5. Handling Uncertain Knowledge, Radom Variables, Prior and Posterior Probability, Inference using Full Joint Distribution, Bayes' Rule and its use, Bayesian Networks, Reasoning in Belief Networks

4.6. Fuzzy Logic

==========