4 CHAPTER

KNOWLEDGE REPRESENTATION

4.0 INTRODUCTION

Knowledge is a progression that starts with data which is of limited utility. Data when processed becomes information, information when interpreted or evaluated becomes knowledge and understanding of the principles embodied with the knowledge is wisdom. This is shown in Fig. 4.1 in knowledge, lies the power.

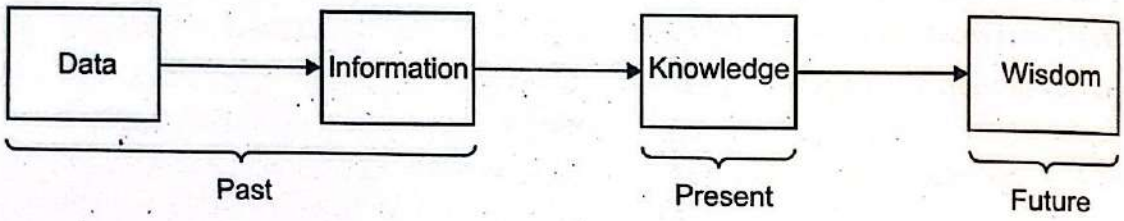


Fig. 4.1 Relationship between data, information, knowledge and wisdom.

A good knowledge representation system should possess the following properties:-

1. **Representational adequacy:** It is defined as an ability to represent the required knowledge.
2. **Inferential adequacy:** It is defined as an ability to manipulate the knowledge represented to produce new knowledge corresponding to that inferred from the original.
3. **Inferential Efficiency:** It is defined as an ability to direct the inferential mechanisms into the most productive directions by string appropriate guides.
4. **Acquisitioned Efficiency:** It is defined as an ability to acquire new knowledge using automatic methods where possible rather than reliance on human intervention.

Please understand that information is static while knowledge is dynamic as it lies within us. Even our expert systems (to be discussed later) are **knowledge-based systems**. These systems are used by **knowledge engineers.** and the process of development building and maintaining knowledge based systems is the domain of **Knowledge Engineering. Knowledge is central to A.I. An A.I.** system must be capable of doing three things:

(*a*) Store knowledge.

(*b*) Apply the knowledge stored to solve problems.

(*c*) Acquire new knowledge through experience.

Also note that any AI system has 3 key components Representation, Reasoning and Learning.

In A.I. different types of knowledge needs to be represented.

(1) **Objects:** Facts about objects in our world domain need to be represented

For example: Guitars have strings. The fact associated with the object guitar is that they have strings. We need to store and represent this fact.

(2) **Events:** Actions that occur in real world.
For example: Arjun played guitar in boys band group.

(3) **Performance:** A behavior. For example, playing a guitar involves knowledge about how to do things.

(4) **Meta-knowledge:** It is the knowledge about what we know.
For example: A Robot to manage your moods.

So to solve problems in A.I. we must represent knowledge. There are two entities involved here.

(*a*) **Facts:** Are data or instance that are specific and unique. These are the truths about the real world. It is also considered as a **knowledge level.**

(*b*) **Representation of facts:** We usually define the representation in terms of symbols that can be manipulated by programs. It is also called as a **symbol level.**

Knowledge Types

Major classification of knowledge is as follows:

Tacit (or procedural) knowledge and Explicit or declarative knowledge

Let us tabulate the differences between them now.

Tacit Knowledge or Procedural	Explicit Knowledge or Declarative
1. It is an *embodied* knowledge that exists *within* a human being.	1. It is an *embedded* knowledge that exists *outside* the human being.
2. It is difficult to articulate formally.	2. It can be easily articulated formally.
3. It is difficult to share or communicate this knowledge.	3. This knowledge can be shared, copied, processed and stored.
4. This knowledge is drawn from experience, action, subjective insight.	4. This knowledge is drawn from artifact of some type as principle, procedure, process and concepts.
5. It is a procedural knowledge about how to do something.	5. It is a declarative knowledge that something is true or false.
6. It focuses on tasks that must be performed to reach on particular objective and goal.	6. It refers to the representations of objects and events, knowledge about facts and relationships.
Examples: Procedures, rules, strategies, agendas, models.	**Examples:** concepts, objects, facts, propositions, assertions, semantic nets, logic and descriptive models.
7. It is hard to debug	7. It is easy to validate.
8. Process oriented.	8. Data oriented.
9. Representations in form of set of rules.	9. Representation in form of production system.

Other sources of knowledge also exist –

(*a*) **Inheritable knowledge**

We can inherit certain type of knowledge also. **For example:**

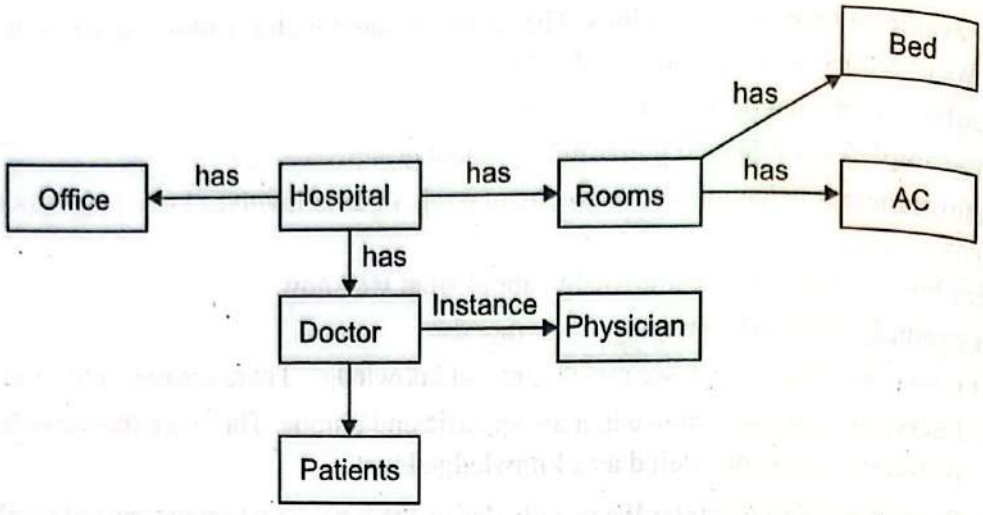


Fig. 4.2 Hospital attribute representation.

The 'hospital' object has certain features like it will have rooms, doctors, office etc. Furthermore, Rooms have bed, AC etc. Doctor has patients and so on. Here, the relationship 'has' indicates the salient features or **property or attributes.** The physician is an instance of Doctor. This is known as **property inheritance** which we shall study a bit later.

(*b*) Inferential knowledge

The knowledge representation method which can use inference mechanism to use this knowledge is called as inferential knowledge. The inferential knowledge requires an inference mechanism for the purpose of exploiting it. Many inference procedures like Resolution will be discussed a bit later. **For example:** Predicate logic.

(*c*) Relations knowledge

This knowledge is represented in of relations/tables in DBMS, form example. the knowledge regarding EMPLOYEE may be represented as

EMPNO	ENAME	EADDR	EPHONE
1.	Vineet	MAIT	852136
2.	Mayur	IITM	526521
3.	Ankit	ITM	921211
4.	Karan	GTBIT	912110
5.	Vijay	VIPS	891929

It can be used to answer simple queries like 'What is the phone number of Vineet'. But this representation cannot store any semantic information like it cannot answer queries like 'Is Vineet a good employee'?.

(*d*) Heuristic knowledge

Heuristics means '**rule of thumb.**' Guessing is the way of making conclusions. It is a form of judgemental information. For example. say, if TV is not working. The guess would be that either tube is not working or probably power is OFF. This is heuristic knowledge.

(*e*) Commonsense knowledge

A knowledge gained by our experience about any general phenomenon.

For example. In order to differentiate human from other animals, the presence of commonsense knowledge plays a vital role.

(*f*) Explicit knowledge

A knowledge which an individual gains explicitly is called as explicit knowledge.

For example: two raised to three is eight ($2^3 = 8$) is a kind of explicit knowledge. This knowledge can be easily communicated, easily processed by computers and sent electronically.

(g) Uncertain knowledge

Today, we are surrounded by many uncertainties. These uncertainties need to be considered as they also provide some sort of knowledge.

For example : If it is cloudy, it will rain.

We use a reasoning process on the available knowledge to draw conclusions. We need such a system which can handle these uncertainties.

Do you know? In human brain, knowledge is stored in form of neurons whereas in computers knowledge is stored as symbolic structures.

Knowledge is a collection of "facts" from some domain. We need a representation of facts that can be manipulated by a program. English cannot be used directly to draw inferences. We need some symbolic representation. **Please note that we must be able to map facts-to-symbols and vice versa using forward and backward representation mapping.** The basic knowledge representation of facts is shown in Fig. 4.3. Consider an English sentence.

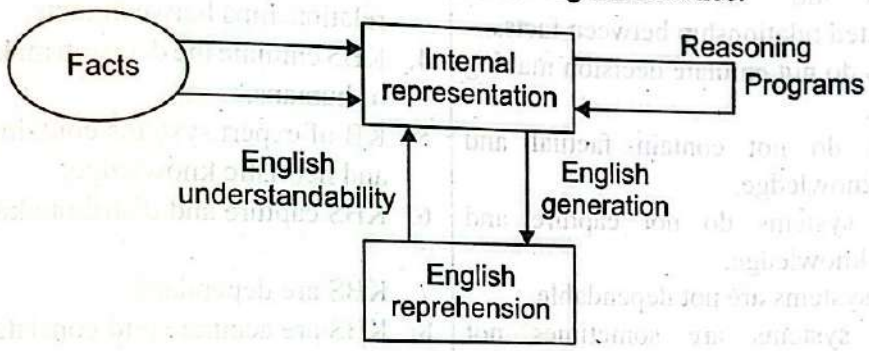


Fig. 4.3 Basic knowledge representation of facts (an English sentence)

For example : consider a 'fact' represented in '**English**' as follows —

'Joe is a father'

Using forward mapping function, the above fact is represented in logic as —

Father (Joe)

On the other hand, **if we use backward mapping function,** then we can generate that English statement back again, i.e., we get –

'Joe is a father'

What are those forward and backward representations?

Forward and Backward representations are shown in Fig. 4.4

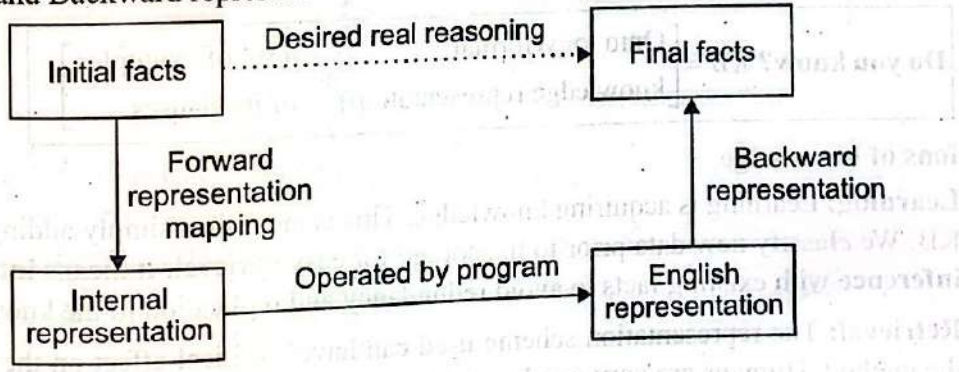


Fig. 4.4 Forward v/s backward representations.

Please note the dotted line on the top. It shows the abstract reasoning process that a program is intended to model. The solid lines on bottom indicate the concrete reasoning process that the program performs.

How to store knowledge?

Collected and stored knowledge is called as **a knowledge base (KB).** A KB is a set of files that stores the knowledge for **knowledge management.** It may be a machine readable as a human readable KB (like articles, papers, user marvels. etc.).

Let us tabulate the differences between database and knowledge base now.

Database (DB)	Knowledge Base(KB)
1. Database stores data that is simple and flat (relational).	1. Knowledge base can be used to store cause and effect information and imprecise and probabilistic information.
2. Database does not consist of fuzzy facts.	2. Knowledge base may consist of fuzzy facts. There is a mechanism which manipulates uncertain information.
3. Databases do not contains more sophisticated relationship between facts.	3. KB contains more sophisticated relationships between facts.
4. Databases do not emulate decision making processes.	4. KBS emulate the decision making processes of humans.
5. Databases do not contain factual and heuristic knowledge.	5. KB of expert systems contains both factual and heuristic knowledge.
6. Database systems do not capture and distribute knowledge.	6. KBS capture and distribute knowledge.
7. Database systems are not dependable.	7. KBS are dependable.
8. Database systems are sometimes not consistent.	8. KBS are accurate and consistent.
9. Database systems are sometimes not profitable.	9. KBS are profitable.
10. A database expert is needed to update databases.	10. An expert knowledge of the domain is needed for updating KB.

Characteristics of a good KB

1. A good KB must be rich in both quality and quantity.
2. It should have carefully written articles.
3. It should be updated regularly.
4. It should have an excellent search engine for easier accessibility.
5. It should have a very good design of the content format.

Do you know? $KB = \begin{bmatrix} \text{Onto logy(formal} \\ \text{knowledge representation)} \end{bmatrix}^{+} \begin{bmatrix} \text{a set of examples} \\ \text{of its clauses} \end{bmatrix}$

Applications of knowledge

1. **Learning:** Learning is acquiring knowledge. This is more than simply adding new facts to KB. We *classify* new data prior to its storage for easy retrieval. It means **interaction and inference** with existing facts to avoid redundancy and replication in the knowledge.
2. **Retrieval:** The representation scheme used can have a critical effect on the **efficiency** of the method. Humans are very good at it. Many AI methods have tried to modal a human.

3. **Reasoning:** It means to infer facts from existing data.

4.1 PROPOSITIONAL LOGIC AND ITS RESOLUTION

A logic is defined as a formal system in which the formulae or sentences have true or false values. In 1976, Robert Kowalski came up with an equation

Algorithm = Logic + Control

The **logic component** specifies the knowledge to be used in solving problems.

The **control component** determines the problem solving strategies by means of which that knowledge is used.

Please note that the logic component determines the meaning of the algorithm whereas the control component only affects its efficiency. In reasoning about truth values, we use a number of operators like

Operator	Meaning
$\wedge$	and
$\vee$	or
$\neg$	not
$\rightarrow$	implies
$\leftrightarrow$	iff (if and only if)

It is useful to represent these logical operators using truth table showing the possible values that can be generated by applying an operator to truth tables.

Let us now see various truth tables (for these operators).

1. **Not ($\neg$) operator:** It is a **unary operator** i.e. it is applied only to one variable

   Its truth table is as follows —

A	$\neg A$
true	false
false	true

2. **and ($\wedge$) operator:** It is a **binary operator,** i.e. it acts on two variables. It is also called as **conjunctive operator.** Its truth table is as follows:

A	B	$A \wedge B$
false	false	false
false	true	false
true	false	false
true	true	true

NOTE: that $A \wedge B$ is true if both A and B are true else it is False. A and B can be any statement or proposition.

3. **or ($\vee$) Operator:** It is again a **binary operator** and is also known as **disjunctive operator.**

   Its truth table is a follows –

A	B	$A \vee B$
false	false	false
false	true	true
true	false	true
true	true	true

So, $A \vee B$ is true for all cases except for $A = B$ = false. This table represents an **inclusive or operator**. An exclusive or would have been 'false' in the final row.

4. Implies ($\rightarrow$) Operator: Its truth table is as follows –

A	B	$A \rightarrow B$
false	false	true
false	true	true
true	false	false
true	true	true

This type of implication is also called as **material implication**. Also note that in statement,

$A \rightarrow B$

'A' is known as **antecedent.**

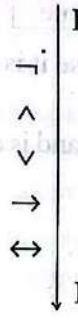
and 'B' is known as **consequent.**

$A \rightarrow B$ is real as 'A implies B' or 'if A then B' or 'if A is to B then B is true'

We can also construct complex truth tables like for $A \vee (B \vee C)$ the truth table is as follows—

A	B	C	$A \wedge (B \vee C)$
false	false	false	false
false	false	true	false
false	true	false	false
false	true	true	false
true	false	false	false
true	false	true	true
true	true	false	true
true	true	true	true

for $n = 3$ (i.e., 3 variables A, B and C) these are 8 outputs. **In general, for n inputs there will be 2^n outputs. Please note that the use of brackets is very important because $A \wedge (B \vee C) \neq (A \wedge B) \vee C$. Also note that to avoid ambiguity, these logical operators are assigned precedence.** The order of precedence is as follows –



This means that $\neg A \vee B = (\neg A) \vee B$

Tip: Use braces whenever an expression becomes ambiguous.

5. **Tautology:** Consider the following truth table –

A	$AV \neg A$
false	true
true	true

This truth table has a property, that is, the value of expression is always true irrespective of the value of A. **Such an expression that is always true is known as tautology.** If A is a tautology, we write —

$|= A$

A logical expression that is tautology is often described as being valid. A **Valid expression** is defined as being one that is true under any interpretation. In other words, no matter what meanings and values we assign to variables in a valid expression, it will still be true. If an expression is false in any interpretation, it is described as being contradictory. The following expressions are contradictory:

(*a*) $A \wedge \neg A$

(*b*) $(A \vee \neg A) \rightarrow (A \wedge \neg A)$

It does not matter what A means in these expressions. The result cannot be time.

Some expressions are **satisfactory** but **not valid.** This means that they are true under some interpretation but not under all interpretations. The following expressions are satisfactory —

(*a*) $A \vee A$

(*b*) $(A \wedge B \vee \neg C) \rightarrow (D \wedge E)$

Please note that a contradictory expression is clearly not satisfactory and so is described as being unsatisfactory.

Tip: Logic is studied as knowledge representation language in A.I.

Logics are of different types —

(*a*) Propositional logic.

(*b*) Predicate logic.

(*c*) Temporal logic.

(*d*) Modal logic.

(*e*) Description logic. etc.

Out of these propositional logic and predicate logic are fundamental to all logic.

Propositional logic

Propositions are statements used in mathematics. A proposition or sentence is classified as declarative sentence whose value is either 'true' or 'false'.

For example. —

(*a*) The sky is blue.

(*b*) Snow is white.

(*c*) 4 * 4 = 16

Propositions are of two types —

(*a*) Atomic propositions.

(*b*) Molecular propositions.

Atomic propositions are single propositions. The above given examples are all propositions.

Molecular propositions are formed by combining two or more atomic propositions.

For example. —

"Lata Mangeshkar is a singer and Rajiv Chopra is an author".

Propositions are 'sentences' either true or false but not both. A sentence is the smallest unit in propositional logic. If proposition is 'time', then truth value is 'time'. If proposition is 'false' then truth value is 'false'.

NOTE: Propositional logic is also called as Propositional Calculus or Sentential Calculus or Boolean Algebra.

Please understand that propositional logic tells the ways of joining and/or modifying entire propositions, statements or sentences to form more complicated propositions, statement or sentences as well as the logical relationships and properties that are derived from the methods of combining or altering statements. Also note that the process used for performing manipulation of the symbols is according to some rules and laws.

A BNF (Backus Naur Form) grammar of Sentences in Propositional logic is as follows —

Sentence → Atomic sentence |complex sentence|

Atomic sentence → true |false| symbol

Symbol → $P|Q|R\ldots$

Complex sentence → ¬ sentence

| (Sentence ∧ Sentence)

| (Sentence □ Sentence)

| (Sentence → Sentence)

| (Sentence ↔ Sentence)

Some terminologies in Propositional Logic

1. **Statement:** A statement is a sentence that is TRUE or FALSE. These statements have some properties:–

   (*a*) **Satisfiability:** A statement is satisfiable if there is some interpretation for which it is true.

   (*b*) **Contradiction:** A sentence is contradictory (unsatisfiable) if there is no interpretation for which it is true. **For example,** Japan is capital of India.

   (*c*) **Validity:** A sentence is valid if it is true for every interpretation.

   For example, the sentence $P \vee \neg P$ is valid. Valid sentences are also called as tautologies.

   (*d*) **Logical Equivalence:** Two sentences are logically equivalent if they have the same truth table under every interpretation, written as $P \leftrightarrow Q$.

   For example, $P \wedge Q$ and $Q \wedge P$ are logically equivalent and this can be shown using truth tables given earlier.

2. **Connective or operator:** The connectives join simple statements into compound statements and joins compound into larger compound statements.

For example : 6 basic connectives and their symbols

Connectives	Symbols	Meaning
Assertion	P	'P is true'
Negation	$\neg P$, ~, !, not	'P is false'
Conjunction	$p \wedge g$, AND, &	'Both p and q are true'
Disjunction	$p \vee q$, i, OR	'Either p is true or q is true or both'
Implication	$p \rightarrow q$, $\supset$, $\Rightarrow$, if then	'if p is true from q is also true' (i.e., p implies q)
Equivalence	$\leftrightarrow$, $\equiv$, $\Leftrightarrow$, if and only if	'P and q are either both true or both false'

NOTE: Both propositions and connective are the basic elements of propositional logic.

3. **Truth value:** The truth value of a statement is its TRUE or False value.

   For example

   (*a*) q is either TRUE or FALSE.

   (*b*) $\sim q$ is either TRUE or FALSE.

   (*c*) $q \rightarrow p$ is either TRUE or FALSE.

NOTE: Use 'T' or '1' to mean TRUE & 'F' or '0' to mean FALSE

For example.

p	q	$\neg p$	$\neg q$	$p \vee q$	$p \rightarrow q$
T	T	F	F	T	T
T	F	F	T	T	F
F	T	T	F	T	T
F	F	T	T	F	T

4. **Tautologies:** A proposition that is always true is called as a tautology.

   For example: $(p \vee \neg p)$ is always true irrespective of the value of p.

5. **Contradictions:** A proposition that is always false is called as a contradiction.

   For example: $(q \vee \neg g)$ is always false regardless of the truth value of the proposition.

6. **Contigencies:** A proposition is called as a contigency, if that proposition is neither a tautology nor a contradiction.

   For example : $(p \vee q)$ is a contingency

7. **Antecedent and consequent:** In the conditional statements, $p \rightarrow q$, 'p' is the 'if-clause' and is called as **antecedent** whereas 'a' is the 'then clause' and is known as **consequent.**

8. **Argument:** Any argument can be expressed as a **compound statement. If we take all premises, conjoin them and make that conjunction the antecedent of a conditional and make the conclusion the consequent then this implication statement is called as the corresponding conditional of the argument.**

Please remember the following points here:

1. Every **argument** has a **corresponding condition.**
2. Every **implication statement** has a **corresponding argument.**

3. Because the corresponding condition of an argument is a statement, it is therefore either a **tautology or a contradiction or a contingency.**
4. An argument is **valid** "if and only if" its corresponding conditional is a **tautology.**
5. Two statements are **consistent** 'if and only if' their **conjunction is not a contradiction.**
6. Two statements are **logically equivalent** "if and only if" their truth table columns are identical or "if and only if" the statement of their equivalence using "≡" is a **tautology.**

NOTE: Truth tables are used to test validity tautology, contradiction, contingency. consistency and logical equivalence.

A logical system is defined in terms of its syntax, its semantics and a set of rules of deduction for proving.

An expression is referred to as a well formed formula (WFF) if it is constructed correctly according to the rules of the syntax of propositional calculus. We define a sentence recursively in terms of other sentences.

More formally **a WFF consists of atomic symbols joined with connectives.**

$\therefore P, P \wedge \neg P, P \wedge Q, P \vee Q, P \rightarrow Q, P \leftrightarrow Q$ are WFFs.

A WFF is defined recursively as —

1. An atom (say P) is a WFF.
2. If P is a WFF then $\neg p$ *is a WFF.*
3. If P and Q are WFF, then $(P \vee Q)$, $(P \wedge Q)$, $(P \rightarrow Q)$ and $(P \leftrightarrow Q)$ are WFF.
4. A string of symbols is a WFF if and only if it is obtained by using above rule 1 to rule 3

Also note the following points about WFF:

1. A WFF is not a proposition but if we substitute a proposition in place of a propositional variable, we get a proposition.

   For example : $\neg (P \vee Q) \wedge (Q \wedge R) \rightarrow (\neg Q \wedge R)$
2. The WFF is also simply called as a formula.
3. WFF are conveniently represented as **truth tables.**
4. A WFF for all true entries is called as a tautology.
5. Two WFFs, say α and β in propositional variables $p_1, p_2 \cdots p_n$ are equivalent if the formula $\alpha \rightarrow \beta$ is a tautology then, the equivalent statements are represented as $\alpha \equiv \beta$.
6. **Please understand that $\alpha \leftrightarrow \beta$ and $\alpha \equiv \beta$ are different as $\alpha \leftrightarrow \beta$ is a formula** whereas $\alpha \equiv \beta$ is not a formula but a relation between α and β.

The semantics (meaning) of the operators of propositional calculus can be defined in terms of truth tables. If our knowledge is represented in form of propositional logic then it is very easy for a computer to do reasoning on this knowledge. The set of inference rules and laws that are used for reasoning are given below:

Rule 1: Idempotency Rule

(*a*) $P \vee P = P$

(*b*) $P \wedge P = P$

Rule 2: Commutative Law

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

(b) $P \wedge Q = Q \wedge P$

(c) $P \leftrightarrow Q = Q \leftrightarrow P$

Rule 3: Associative Law

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

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

Rule 4: Distributive Law

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

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

Rule 5: De Morgan's rule

(a) $\sim (P \vee Q) = \sim P \wedge \sim Q$

(b) $\sim (P \wedge Q) = \sim P \vee \sim Q$

Rule 6: Implication Removal

$P \leftrightarrow Q = \sim P \vee Q$

Rule 7: Biconditional elimination

$P \leftrightarrow Q = (P \rightarrow Q) \wedge (Q \rightarrow P)$

Rule 8: Absorption Law

(a) $P \vee (P \wedge Q) \equiv P$

(b) $P \wedge (P \vee Q) \equiv P$

Rule 9: Contrapositive

$P \Rightarrow Q \equiv \neg Q \Rightarrow \neg P$

Rule 10: Double negation

$P \equiv \neg (\neg P)$

Rule 11: Fundamental identities

(a) $P \vee \neg P \equiv T$

(b) $P \wedge \neg P \equiv F$

(c) $P \vee \neg T \equiv T$

(d) $P \vee T \equiv P$

(e) $P \vee F \equiv P$

(f) $P \vee F \equiv F$

(g) $(F \neg Q) \wedge (P \Rightarrow \neg Q) = \neg P$

(h) $P \Rightarrow Q \equiv (\neg P \vee Q)$

Rule 12: AND ($\wedge$) rule (Introduction)

Given P and Q. We deduce $P \wedge Q$ i.e.

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

This follows from the definition of $\wedge$.

Rule 13: AND ($\wedge$) rule (Elimination)

Given P and Q. We deduce P and Q i.e. separately i.e.

(*a*) $P \wedge Q \rightarrow P$

(*b*) $P \wedge Q \rightarrow Q$

This also follows from the definition of $\wedge$.

Rule 14: OR ($\vee$) rule (Introduction)

Given P and Q, We deduce P and Q i.e. separately i.e.

(*a*) $A \rightarrow A \vee B$

(*b*) $B \rightarrow A \vee B$

i.e. from A we can deduce the disjunction of A which any expression.

Rule 15: Modus Ponens

From given two statements P and $P \rightarrow Q$, infer Q i.e.,

$$\frac{P \quad P \rightarrow Q}{\therefore \; Q}$$

NOTE: In implication form, this rule is written as

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

For example.

Given: Rajiv is intelligent

and: Rajiv is intelligent $\rightarrow$ Rajiv is author.

Conclude: Rajiv is author.

Rule 16: Modus Tollens

From the given two statements, $\neg Q$ and $(P \rightarrow Q)$, infer $\neg P$ i.e.

$$\frac{\neg P \quad P \rightarrow Q}{\therefore \; \neg Q}$$

For example :

Given: Ajay is not a religious person.

and: Ajay goes to temple daily implies Ajay is a religious person

Conclude: Ajay does not go to temple daily.

NOTE: In implication form, this rule is written as

$$(\neg Q \wedge (P \rightarrow Q)) \rightarrow \neg Q$$

Rule 17: Chain Rule/Hypothetical syllogism

From $(P \rightarrow Q)$ and $Q \rightarrow R)$, infer $(P \rightarrow R)$ i.e.

$$\frac{P \rightarrow Q \quad P \rightarrow Q}{\therefore \; (P \rightarrow R)}$$

For example :

Given: It is raining → must take umbrella

and: Must take umbrella → There is no sun

Conclude: There is no sun.

NOTE: In implication form, this rule is written as

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

Rule 18: Reductio Ad Absurdum

This rule states that if we assume that A is false ($\neg A$) and this leads to a contradiction ($\perp$) then we say that A is true. This is known as proof by contradiction. This is

$$\begin{array}{l} \neg A \\ \vdots \\ \perp \\ \hline \therefore A \end{array}$$

Here, the symbol $\perp$ is called as **falsumn** which is used to indicate an absurdity or a contradiction.

For example., $\perp$ can be deduced from $A \wedge \neg A$.

Rule 19: Disjunctive syllogism

From two given sentences, $\neg P$ and $(P \vee Q)$, infer Q i.e.,

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

NOTE: In implication form, this rule is written as

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

Rule 20: Constructive and Destructive dilemma

From given two sentences

$$\begin{array}{l} (P \rightarrow Q) \wedge (R \rightarrow S) \\ \text{and} \qquad (P \vee R) \\ \hline \therefore (Q \vee S) \end{array}$$

(Constructive dilemma)

Also,

$$\begin{array}{l} (P \rightarrow Q) \wedge (R \rightarrow S) \\ \neg (P \vee R) \\ \hline \therefore (Q \vee S) \end{array}$$

(Destructive dilemma)

NOTE: In implication form, this rule is written as

$((P \rightarrow Q) \wedge (R \rightarrow S))\ (\neg Q \wedge S) \rightarrow (P \vee R)$.

We are in a position to solve some examples now.

EXAMPLE 1: Given P and Q, prove that

$((P \rightarrow Q) \rightarrow P) \rightarrow P$ is tautologous.

SOLUTION: We draw its truth table —

P	Q	$((P \rightarrow Q) \rightarrow P) \rightarrow P$		
T	T	T	T	T
T	F	F	T	T
F	T	T	F	T
F	F	F	F	T

Hence, $((P \rightarrow Q) \rightarrow P) \rightarrow P$ is tautologous.

EXAMPLE 2: Find the equivalent expression of

A (& $(A \vee B)$) using truth table

SOLUTION:

A	B	$(A \vee B) = C$ (say)	A & C
t	t	t	t
f	f	f	f
t	f	t	t
f	t	t	f

$\therefore \quad A\,(\&\,(A \vee B)) = A$ $\quad [\because AB' + AB = A\,(B' + B) = A]$

EXAMPLE 3: Prove that $(A \wedge B) \rightarrow A \vee B$.

SOLUTION: Given $A \wedge B$ (assumption)

$\rightarrow A$ (by $\wedge$ elimination)

$\rightarrow A \vee B$ (by $\vee$ introduction)

Hence Proved.

EXAMPLE 4: Prove that —

$((P \wedge Q) \rightarrow R) \vee (\sim Q \rightarrow \sim R)$ is valid.

SOLUTION: Let us denote this statement by Q

Q: $((P \wedge Q) \rightarrow R) \vee (\sim Q \rightarrow \sim R)$

We draw its truth table now:

P	Q	R	$P \wedge Q$	$P \wedge Q \rightarrow R$	$\sim Q \rightarrow \sim R$	Q
T	T	T	T	T	T	T
T	T	F	T	F	T	T
T	F	T	F	T	F	T
T	F	F	F	T	T	T
F	T	T	F	T	T	T
F	T	F	F	T	T	T
F	F	T	F	T	F	T
F	F	F	F	T	T	T

$\therefore$ Q is true under all interpretations.

$\therefore$ It is a valid statement.

EXAMPLE 5: Show that the set of statements "I will be wet if it rains and I go out of the house. It is raining now. I go out of the house. I will not be wet" are inconsistent.

SOLUTION: W : I will be wet.

R : It rains.

G : I go out of the house.

A set of formulae corresponding to given word problem is —

$\therefore$ $\{(R \wedge G) \rightarrow W, R, \sim W\}$

It can be shown using truth table that

$((R \wedge G) \rightarrow W) \wedge R \wedge G \wedge \sim W$ is false under all interpretations and hence is inconsistent.

EXAMPLE 6: Determine whether each of following is

(a) Satisfiable. (b) Contradictory. (c) Valid.

$S_1 : (P \,\&\, Q) \vee \sim (P \,\&\, Q)$

$S_2 : (P \,\&\, Q) \rightarrow (R \vee \sim Q)$

$S_3 : (P \,\&\, Q) \rightarrow (R \vee \sim Q)$

$S_4 : (P \vee Q) \,\&\, (P \vee \sim Q) \vee P$?

SOLUTION: $S_1 : (P \,\&\, Q) \vee \sim (P \,\&\, Q)$

Its truth table is —

G : I go out of the house.

P	Q	$P \,\&\, Q = A$	$\sim (P \,\&\, Q) = B$	$A \vee B$
t	f	f	t	t
f	t	f	t	t
f	f	f	t	t
t	t	t	f	t

$\therefore$ It is satisfiable.

$\therefore$ It is not contradictory

$\therefore$ It is valid as sentence is true for every interpretation.

Consider $S_2 : (P \,\&\, Q) \rightarrow (R \vee \sim Q)$

P	Q	R	$P \,\&\, Q = X$	$\sim Q$	$R \vee \sim Q = Y$	$X \rightarrow Y$
f	f	f	f	t	t	f
f	f	t	f	t	t	f
f	t	f	f	f	f	t
f	t	t	f	f	t	f
t	f	f	f	t	t	f
t	f	t	f	t	t	f
t	t	t	t	f	t	t

$\therefore$ It is satisfiable

It is not contradictory

It is not valid.

$S_3 : (P \& Q) \rightarrow (R \vee \sim Q)$

Its truth table is —

P	Q	R	$P \& Q = X$	$\sim Q$	$R \vee \sim Q = Y$	$X \rightarrow Y$
f	f	f	f	t	t	f
f	f	t	f	t	t	f
f	t	f	f	f	f	t
f	t	t	f	f	t	f
t	f	f	f	t	t	f
t	f	t	f	t	t	f
t	t	f	t	f	f	f
t	t	t	t	f	t	t

∴ It is satisfiable

It is not contradictory

It is not valid.

$S_4 : (P \vee Q) \& (P \vee \sim Q) \vee P$

P	Q	$P \vee Q = X$	$\sim Q$	$P \vee \sim Q = Y$	$Y \vee P = Z$	$X \& Z$
t	t	t	f	t	t	t
f	t	t	f	f	f	f
t	f	t	t	t	t	t
f	f	f	t	t	t	t

∴ It is satisfiable

It is not contradictory

It is not valid.

Normal Forms in Propositional Logic

There are two major normal forms of statements in propositional logic. They are Conjunctive Normal Form (CNF) and Disjunctive Normal Form (DNF).

Any formula can be converted into its normal form by substituting it by its equivalent formulae. Two formulas P and Q are equivalent if and only if the truth values of P and Q are same for all values of P and Q. We have already discussed some widely used equivalent formulae in table III.

A formula P is said to be in CNF if it is of the form

$$P = P_1 \wedge P_2 \wedge P_3, \ldots, \wedge P_n; n \geq 1$$

where each $P_1, P_2, P_3, \ldots, P_n$ is a disjunction of an atom or negation of an atom.

A formula P is said to be in DNF if it has the form

$$P = P_1 \vee P_2 \vee P_3, \ldots, \vee P_n; n \geq 1$$

where each $P_1, P_2, P_3, \ldots, P_n$ is a conjunction of an atom or negation of an atom.

Conversion Procedure to Normal Form

Step 1: Eliminate implication and biconditionals. We use the following laws

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

$$(P \leftrightarrow Q) = (P \rightarrow Q) \wedge (Q \rightarrow P)$$

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

Step 2: Reduce the NOT symbol by the formula $(\neg(\neg P)) = P$ and apply De Morgan's theorem to bring negations before the atoms.

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

Step 3: Use Distributive laws and other equivalent formula given in table III to obtain the normal form

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

Conjunctive Normal Form

The resolution rule applies only to disjunction of literals. So it would seem to be relevant only for knowledge bases and queries consisting of such disjunctions. How can then it lead to a complete inference procedure for all of propositional logic? **The answer is that every sentence of propositional logic is logically equivalent to a conjunction of disjunctions of literals. A sentence expressed as a conjunction of disjunctions of literals is said to be in a conjunctive normal form** (CNF). Every sentence can be transformed into a CNF sentence using the following steps making use of table 6.2

1. Eliminate $\leftrightarrow$ replacing $P \leftrightarrow Q$ with $(P \rightarrow Q) \wedge (Q \rightarrow P)$
2. Eliminate $\rightarrow$ replacing $P \rightarrow Q$ with $\neg P \wedge Q$.
3. CNF requires $\neg$ should appear only in literals, so we move $\neg$ inwards by repreated application of following equivalences to Table 6.2

$$\neg(\neg P) \equiv \text{Double-negation elimination}$$
$$(P \wedge Q) \equiv (\neg P \wedge \neg Q) \text{ de Morgan}$$
$$\neg(P \vee Q) \equiv (\neg P \vee \neg Q) \text{ de Morgan}$$

4. We now apply distributivity law, distributive $\vee$ over $\wedge$. whenever the sentence contains nested $\vee$ over $\wedge$ operators applied to literals.

EXAMPLE 1: Convert $((P \rightarrow Q) \rightarrow R)$ into CNF

SOLUTION: $((P \rightarrow Q) \rightarrow R)$

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

Hence, $(P \vee R) \wedge (\neg Q \vee R)$ is the CNF of $((P \rightarrow Q) \rightarrow R)$

We are in a position to solve some examples now.

EXAMPLE 2: Convert $(P \rightarrow Q) \rightarrow R$ to CNF?

SOLUTION: Given: $(P \rightarrow Q) \rightarrow R$

$$= (\sim P \vee Q) \rightarrow R$$
$$= \sim(\sim P \vee Q) \vee R \qquad [\because P \rightarrow Q = \sim P \vee Q]$$
$$= (P \,\&\, \sim Q) \vee R \qquad \text{[De Morgan's law]}$$
$$= (P \vee R) \,\&\, (\sim Q \vee R) \qquad \text{[By distributive law]}$$

EXAMPLE 3: Convert $\sim(P \& Q) \& (P \vee Q)$ **to DNF?**

SOLUTION: $(\sim P \vee \sim Q) \& (P \vee Q)$ [De Morgan's law]

$= ((\sim P \vee \sim Q) \& P) \vee ((\sim P \vee \sim Q) \& Q)$ [By distributive law]

$= (\sim P \& P) \vee (\sim Q \& P) \vee (\sim P \& Q) \vee (\sim Q \& Q)$

EXAMPLE 4: Convert $\sim(P \vee \sim Q) \& (R \rightarrow S)$ **to DNF?**

SOLUTION: $(\sim P \& Q) \& (\sim R \vee S)$ [De Morgan's law]

$= \sim(\sim((\sim P \& Q) \& (\sim R \vee S)))$ [Negative 2 times]

$= \sim((P \vee \sim Q) \vee (R \& \sim S)$

EXAMPLE 5: Convert $P \rightarrow ((Q \& R) \leftrightarrow S)$ **to DNF?**

SOLUTION: $F \rightarrow G = \sim F \vee G$

$\therefore$ We have —

$= (\sim P) \vee ((Q \& R) \rightarrow S$

$= (\sim P) \vee (\sim (Q \& R) \vee S) \& (\sim S \vee (Q \& R))$ [$\because P \leftrightarrow Q = (\sim P \vee Q) \& (\sim Q \vee P)$]

$= (\sim P) \vee (((\sim Q \vee \sim R) \vee S) \& ((\sim S \vee Q) \& (\sim S \vee R)))$ [Distributive law?]

$= (\sim P) \vee ((\sim Q \vee \sim R) \vee S) \& ((\sim S \vee Q) \& (\sim S \vee R)))$

EXAMPLE 6: Convert $(\sim P \& Q) \vee (P \& \sim Q) \& S$ **to CNF?**

SOLUTION: $(\sim P \& Q) \vee P \& (\sim Q \& S)$ (Associativity)

$= ((\sim P \vee P) \& (Q \vee P) \& (\sim Q \& S))$ [Distributive law]

Resolution in Propositional Logic [By Robinson]

Resolution is the process of producing proofs by refutation or contradiction. It is a procedure used for **theorem proving.** It is the rule of **inference. The procedure of resolution is as follows:**

1. Assume that the negation of the theorem which we want to prove is time.
2. Show that the axioms and the assumed negation of the theorem together cannot be true.
3. Conclude that the assumed negation of the theorem cannot be true because it leads to contradiction.
4. Conclude that the theorem must be true as the assumed negation of the theorem is not true.

Resolution is applicable only on the facts represented in clausal form. A clause is a special formula expressed as disjunction of literals. If a clause contains only one literal then it is called as a **visit clause. Please note that CNF representation of a formula is of the form** $(C_1 \wedge C_2 \wedge \ldots \wedge C_n)$ **where each** $C_k (1 \leq k \leq n)$ **is a clause**

If two clauses C_1 and C_2 contain a complementary pair of literals $\{L, \sim L\}$ then these clauses may be resolved together by deleting L from C_1 and $\sim L$ from C_2 and constructing a new clause by the disjunction of the remaining literals in C_1 and C_2. The new clause thus generated is called as **resolvent of** C_1 and C_2. Here, C_1 and C_2 are called **parents** of resolved clause. The parents can

contain more than one set of complementary pair of literals. All complementary pair of literals are removed and resolvent is constructed, by the disjunction of the remaining literals in the parents.

For example : Consider two clauses

C_1 with literal $L1$ and $\sim L2$

C_2 with literal $\sim L1$ and $L2$

&

$C_1 = L1 \vee \sim L2 \vee C_1'$

i.e.,

$C_2 = \sim L1 \vee \sim L2 \vee C_2'$

&

where C_1' and C_2' are disjunction of remaining literals in C_1 and C_2 respectively

After resolving C_1 and C_2 we get

An inverted binary tree is generated with the last node of the binary tree to be a **resolvent. This is known as a Resolution tree. Resolved literals are underlined for more clarity.**

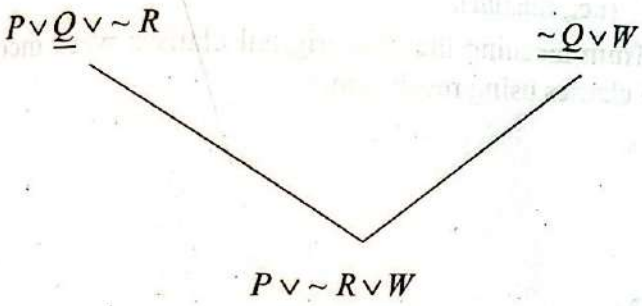
For example : Consider two clauses as

$$C_1 = P \vee Q \vee \sim R$$

$$C_2 = \sim Q \vee W$$

&

$\therefore$



$\therefore$ Resolvent $(P \vee Q \vee \sim R, \sim Q \vee W) = P \vee \sim R \vee W$

Now, consider another example with three clauses —

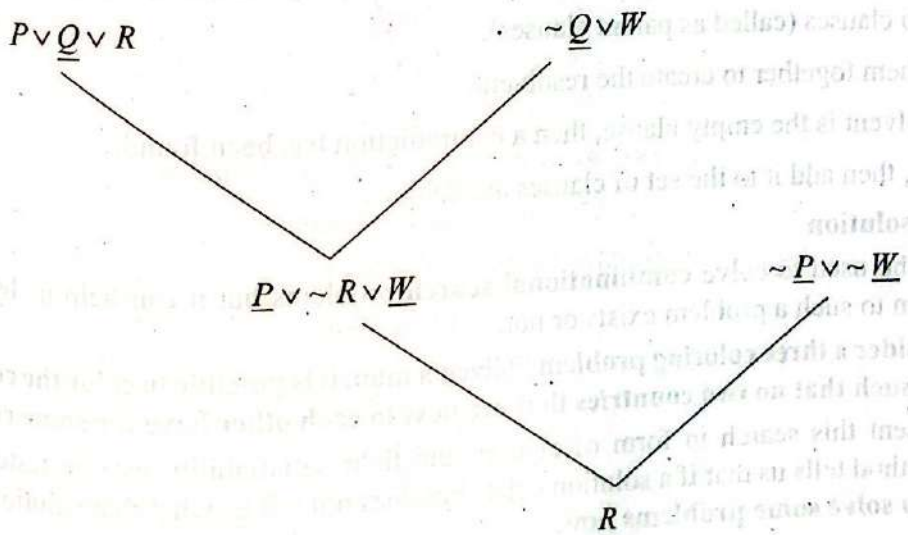
$$C_1 = P \vee Q \vee R$$

$$C_2 = \sim Q \vee W$$

and

$$C_3 = \sim P \vee \sim W$$

$\therefore$



∴ Resolvent $R = (C_1, C_2, C_3)$

Please note that the order of clauses taken for resolution is immaterial, we have to identify two clauses with complementary literals and resolve them. Take the current resolvent and try to find another clause yet not been resolved having complementary literal proceed till all the clauses have been resolved. *Also note that the resolved clauses are not used in resolution process further.*

In a similar manner, in the big system, where there are multiple clauses, all pairs of resolvable clauses are identified and resolved.

Resolution Rule: The resolution of clauses

$(A \vee B)$ and $(\neg B \vee C)$ is $A \vee C$.

Resolution works on the principle of **refutation,** i.e., it presumes that the statement to be proved is not true and then it proves that whatever we have assumed is not true and hence the original statement is true. Now let us resolve the following clause —

$\{(\neg A, B) (\neg A, \neg B, C), A \neg C\}$

We start by resolving the first clause with the second clause, thus eliminating B and $\neg B$ to get

$\{(\neg A, C), A, \neg C\}$

$= \{C, \neg C\}$

$= \perp$ (i.e., falsum).

∴ This resolution has resulted in a **falsum** meaning that the original clauses were inconsistent. Thus, we have refutated the original clauses using resolution.

Refutation (RR)

∴ We say that

$\{(\neg A, B), (\neg A, \neg B, C), A \neg C\} = \perp$

after *RR*.

This is how refutation helps in logical inferencing.

Let us now write an algorithm for Resolution using Propositional Logic —

Algorithm : Resolve (P, F)

Step 1: Convert all propositions of F to clausal form.

Step 2: Negate P and convert the result to clausal form. Add it to the set of clauses obtained in step 1.

Step 3: Repeat until either of the contradictions is found or no progress can be made.

(*a*) Select two clauses (called as parent clauses).

(*b*) Resolve them together to create the resolvent.

(*c*) If the resolvent is the empty clause, then a contradiction has been found.

If it is not, then add it to the set of clauses available.

Application of Resolution

Resolution cannot be used to solve **combinational search** problems but it can help us by telling whether the solution to such a problem exists or not.

For example.: **Consider a three coloring problem "Given a map, it is possible to color the countries using three colors such that no two countries that are next to each other have the some color."**

We can represent this search in form of clauses and their satisfiability can be tested using resolution. This method tells us that if a solution exists but does not tell us what that solution is. **We are in a position to solve some problems now.**

EXAMPLE 1: "If it is hot, then it is humid. If it is humid then it will rain. It is hot." Show that "It will rain".

SOLUTION: Let us denote these statements

H : It is humid

R : It will rain.

O : It is hot.

Formulae corresponding to given sentences are —

∴ If it is hot, then it is humid i.e., $O \rightarrow \mathrm{H} \cong \sim O \vee H$

If it is humid, then it will rain i.e., $H \rightarrow \mathrm{R} \cong \sim Q \vee R$

Construct a set, S, of clauses from above set of statements as given below —

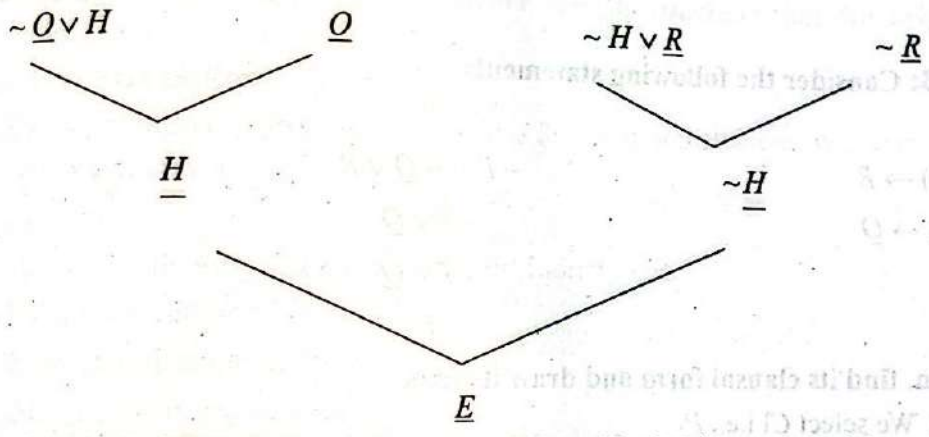
$$S = \{\sim O \vee H, \sim H \vee R, O\}$$

Include negation of "It will rain (say R)" to the set, S.

∴ $$S' = \{\sim O \vee H, \sim H \vee R, O, \sim R\}$$

Now, we can show that S' *is* unsatisfiable, i.e., from S' one can deduce empty clause, then we can infer or conclude that 'It will rain'.

∴ The resolution tree is —



∴ an empty clause (E) is deduced from a set, S',"It will rain" is concluded from S.

EXAMPLE 2: Consider the following statements —

$$A \rightarrow B$$
$$B \rightarrow C$$
$$C \rightarrow D$$
$$\mathrm{D} \rightarrow EVF$$
$$A \rightarrow F.$$

Using resolution, find its clausal from and draw its tree.

SOLUTION: Firstly, we negate the conclusion

$A \rightarrow F$, i.e., we get $\neg(\mathrm{A} \rightarrow F)$.

Next, $D \rightarrow E \vee F$

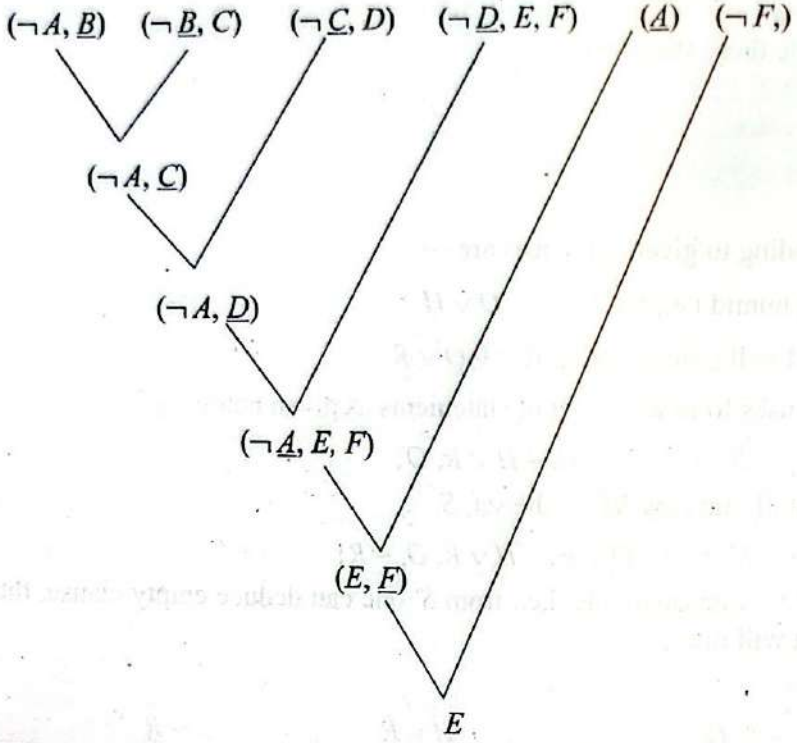
$$\equiv \neg D \vee (E \vee F)$$

& $$\neg(A \rightarrow F) \equiv \neg(\neg A \vee F)$$

$$\equiv A \wedge \neg F$$

∴ Our clauses are, $\mathrm{S} = \{(\neg A, B), (\neg B, C), (\neg C, D), (\neg D, E, F), (A \neg F)\}$

and our proof in tree form will be —



EXAMPLE 3: Consider the following statements

1. P
2. $(P \wedge Q) \rightarrow R$
3. $(S \vee T) \rightarrow Q$
4.
5. T

P

$\sim P \vee \sim Q \vee R$

$\sim S \vee Q$

$\sim T \vee Q$

T

Using resolution, find its clausal form and draw its tree.

SOLUTION: We select $C1$ i.e., P

and then negate $\sim R$.

Now select ($C2$) clauses as

$\sim P \vee \sim Q \vee R$.

$\therefore$ We have two clauses as—

$C1 : \sim R$

$C2 : \sim P \vee \sim Q \vee R$

$\therefore$ Resolvent will be $\sim P \vee \sim Q$.

Here, we have proved R. We should always select those pairs that can be resolved very easily. Now, prove P. We negate $\sim P$ to P.

$C1 : \sim P \vee \sim Q$

$C2 : P$

Resolvent is $\sim Q$.

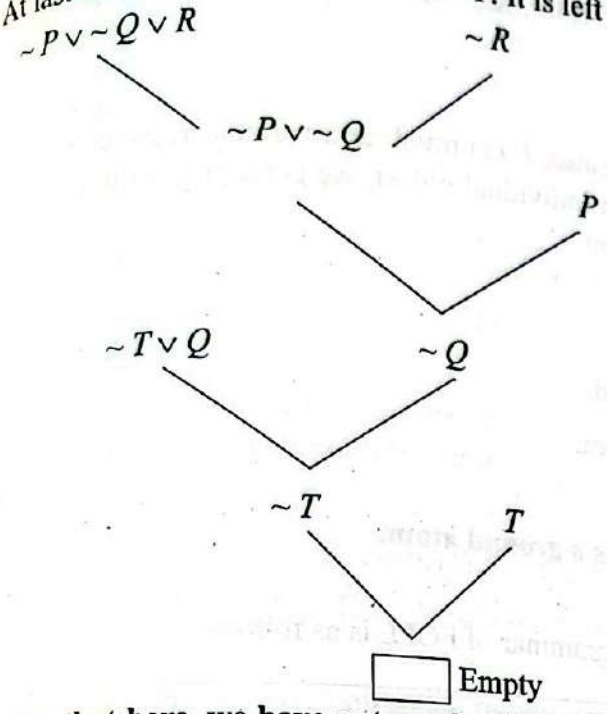
Now prove T. Select $\sim T \vee R$

$C1 : \sim Q$

$C2 : \sim T \vee Q$

$\therefore$ Resolvent is $\sim T$.

At last we have $C1$ as T and $C2$ as $\sim T$. It is left empty which is a contradiction.



Note that here, we have not reached falsum as we are left with a single clause {E} which cannot be resolved with anything now. Hence, we can conclude that our original conclusion was not valid.

Limitations of Propositional Logic

A very serious limitation of propositional logic is that common elements within propositions cannot be used to make inference.

For example.,

Consider the following statements in propositional logic.

S_1 : Rajiv is an author of 12 books.

S_2 : Rajiv is assistant professor.

S_3 : All professors are authors

Now, from S_1, S_2 and S_3 we cannot conclude that all professors are authors because this method of representation fails in capturing the relationship between any individual. Propositional logic lacks in expressiveness which is very vital for good KR. Propositional logic is too coarse to easily describe properties of objects. We cannot make generalized statements.

So, we go to predicate logic.

4.2 PREDICATE LOGIC & ITS RESOLUTION

We are interested in a form of predicate logic called as **first order predicate logic.** FOPL allows the quantified variables to refer to object domain and not to predicates or functions. If the quantification is over first order predicates and functions then it becomes **second order predicate.** Similarly, third order predicate allows quantification over predicates and function of second order and so on. A.I. researchers have found that FOPL is most suitable for KR.

Every complete sentence contains two parts — **a subject** and a **predicate.**

Subject: The subject is what (or whom) the sentence is about.

Predicate: It tells something about the subject.

For example. In the sentence, 'Rajiv is an author' subject is 'Rajiv' and predicate is 'author'.

Sentences involving the predicates that describe the property of objects are denoted by $P(x)$ where

P — The predicate,

x — is a variable denoting any object.

Here, $P(x)$ is not a proposition. This is because $P(x)$ involves a variable x, we cannot assign a truth value to $P(x)$. But if we replace x by an individual object, we get a proposition, like if we replace x by "Rajiv" in $P(x)$ we get a proposition.

Characteristics of predicate logic

1. Logical inferencing is allowed.
2. More accurate *KR* of facts of real world.
3. Program designing is its application area.
4. Better theoretical foundation.
5. A predicate with no variable is called as a **ground atom.**

Syntax of FOPL

A complete description of syntax from the grammar of FOPL is as follows:

Sentence → Atomic Sentence
| (Sentence Connective Sentence)
| (Quantifier variable … Sentence)
| (¬sentence)
Atomic Sentence → Predicate (Term…) | Term = Term
Term → Function (Term…)
| Constant
| Variable
Connective → ⇒ | ∧ | ∨ | ⇒
Quantifier → ∀ | ∃
Constant → a | X_1 | John | …
Variable → a | x | s |
Predicate → Before | HasColour | Raining | …
Function → Mother | LeftLeg

NOTE: Predicate calculus symbols are irreducible syntactic elements just as we have tokens in programming language.

Rules for predicate calculus symbols —

1. Set of letters (uppercase or lowercase) is allowed.
2. Set of digits (0 to 9) is allowed.
3. Underscore (_) is allowed.

But,

1. Blanks and non-alphanumeric characters cannot be used.
2. Special characters like $, *, #, /, are not allowed.

For example.

Valid symbols	Invalid symbols
white	2black
son-of	Ab % b
father-of	**3
Rajiv	— parrot

NOTE: Parentheses, commas and periods are used to construct well formed structure but do not denote objects or relations in the world. These are known as **improper symbols.**

Symbols used

1. **Predicate symbols:** They denote relations or functional mapping from the elements of a domain to the values true or false

   For example.

   Brother, king, LOVE, GREATER, EQUAL, MARRIES etc.

2. **Function symbols:** They denote relations defined on a domain. **For example.** Father of, Age of, plus etc.

3. **Variable symbols:** Lowercase unsubscripted or subscripted letters like x, y, z, t, u, v etc. They can assume different values over a given domain.

4. **Constants:** They are fixed value terms. They are individual symbols which are names of objects like john, rajiv, 1, 2, 3, a, b, c etc.

5. **Quantifiers:** They are of two types —

   (*a*) $\exists$ **(or existential quantifier)** where ($\exists x$) means for some x or there is no x

   (*b*) $\forall$ **(or universal quantifier)** where ($\forall x$) means for all x.

   We will discuss about it a bit later.

6. **Logical Operators / Connectives:** FOPL uses the same five connectives of PL i.e. $\sim$ (not), $\wedge$ (and), $\vee$ (or), $\rightarrow$ (implication) and $\leftrightarrow$ (equivalence).

   Please note that function and predicate symbols take a specified number of arguments. If they take n arguments then it is called as n-place function and predicate respectively. Also note that constants, variables and functions are referred to as terms.

   Predicates are referred to as atomic formulas or atoms.

Precedence of operators & quantifiers

$\sim$ ((Higher priority)
$\forall$ ↓
$\exists$ ↓
$\wedge$ ↓
$\vee$ ↓
$\rightarrow$ ↓
$\leftrightarrow$ (Lower priority)

Quantifiers $\{\forall, \exists\}$ have same priorities as $\sim$ and parentheses have highest priorities.

Using quantifiers

(*a*) **Universal quantifier ($\forall$)**

It is used to represent the phrase **'for all'**. It says that something is true for all possible values of a variable.

For example. (*a*) John loves everyone

In predicate calculus, we write it as

$(\forall x)$ *LOVE* (john, x)

(*b*) for all $n, x^n = x \cdot x \cdot x \ldots x$-times is written as $\forall x\ \forall n\ Q(x, n)$ where

(*b*) **Existential quantifier ($\exists x$):**

It is used to represent the Phrase **'there exist.'** It indicates that the variable is time for at least one interpretation.

For example. (*a*) "There exists x such that $x^3 = 8$." is written as —

$\exists x$: R(x). Where R(x) is $x^3 = 8$.

(*b*) Lord Ram has a crown on his head.

$\exists x$ Crown (x) $\wedge$ On head (x, Ram)

NOTE: ($\forall$) uses $\rightarrow$ connective whereas $\exists$ uses $\wedge$ connective

(*c*) **Nested quantifiers:**

We can use both $\forall$ and $\exists$ quantifiers separately.

For example. 'Brothers are siblings' can be written as —

$\forall x\ \forall y$ Brother (x, y) $\rightarrow$ Sibling (x, y)

or $\forall x, y$ sibling (x, y) $\leftrightarrow$ sibling (y, x)

Example 2: 'Everybody loves somebody' can be written as —

$\forall x\ fy$ loves (x, y)

or $Q(x, n) = x^n$.

Connection between $\forall$ and $\exists$

The two quantifiers are actually intimately connected to each other, through negation. Saying that everyone dislikes garlic is the same as saying that there does not exist someone who likes the garlic:

$\forall x \neg$ likes (x Garlic) $= \neg \exists x$ likes (x, Garlic)

We can go one step further: "Everyone likes ice cream" means that there is no one who does not like ice cream:

$\forall x$ likes (x, Ice Cream) $= \neg \exists x$ likes (x, Ice Cream)

Because $\forall$ is really a conjunction over the universe of objects and $\exists$ is a disjunction, it should not be surprising that they obey de Morgan's rules. The de Morgan rules for quantified and unquantified sentences are:

1. $\forall x \neg P \equiv \neg \exists x P$
2. $\neg \forall x P \equiv \exists x \neg P$
3. $\forall x P \equiv \neg \exists x \neg P$
4. $\exists x P \equiv \neg \forall x \neg P$
5. $\neg P \wedge \neg Q \equiv \neg (P \vee Q)$
6. $\neg (P \wedge Q) \equiv (\neg P \vee \neg Q)$
7. $P \wedge Q \equiv \neg (\neg P \vee \neg Q)$
8. $P \vee Q \equiv \neg (\neg P \wedge \neg Q)$

Thus, we do not really need both $\forall$ and $\exists$ as we do not really need both $\wedge$ and $\vee$.

The range over which a variable is valid is called its scope.

For example. $(\forall x)$ (MAN (x) $\rightarrow$ Mortal (x))

Here, scope of ($\forall x$) is (MAN (x) $\rightarrow$ Mortal (x))

An occurrence of a variable x in a formula is said to be **bound** if and only if the occurrence is within the scope of a quantifier employing the variable x, else a variable is said to be **free.**

For example. In $(\forall x)\, P(x, y)$

x is bound

and y is free

Example 2: In $(\exists x)(\forall y)\, P(x, y) \wedge Q(x)$

x and y are bound in $P(x, y)$ x is free in $Q(x)$ as $Q(x)$ is outside the scope of $(\exists x)$ and $(\forall y)$.

A formula is said to be **closed** if there are no free occurrence of any variable in it.

For example. $(\exists x)(\forall y)(P(x, y) \wedge Q(x))$ is a closed formula

whereas $(\forall y)\, P(x, y)$ is not.

We are in a position to solve some problems now.

EXAMPLE 1: Represent the following facts in predicate logic —

(*a*) **Snow is white.**

(*b*) **Marcus was a man.**

(*c*) **Marcus was a Pompeian.**

(*d*) **All Pompeians were Romans.**

(*e*) **Caesar was a ruler.**

(*f*) **All Romans were either loyal to Caesar or hated him.**

(*g*) **Everyone is legal to someone.**

(*h*) **Marcus tried to assassinate Caesar.**

(*i*) **If it rains then sky will be cloudy.**

(*j*) **If you will not work hard, you will fail.**

SOLUTION: (*a*) Snow (white).

(*b*) Man (Marcus).

(*c*) Pompeian (Marcus)

(*d*) $\forall x$: Pompeian $(x) \rightarrow$ Roman (x)

(*e*) Ruler (Caesar)

(*f*) $\forall x$: Roman $(x) \rightarrow$ loyal to $(x$, Caesar) $\vee$ hate x, (Caesar)

(*g*) $\forall x\, \exists y$ loyalto (x, y)

(*h*) tryassassinate (Marcus, Caesar).

(*i*) raining $\rightarrow$ cloudy (sky)

(*j*) $\neg$ Workhard $\rightarrow$ fail

EXAMPLE 2: Write the predicate forms of the following —

(*a*) **All indoor games are easy.**

(*b*) **Rajiv only likes cricket game.**

(*c*) **All dogs are mammals.**

(*d*) **Roses are red.**

(*e*) **John is father of Bob.**

SOLUTION: (*a*) $\forall x$ indoor game $(x) \rightarrow$ easy (x)

(*b*) likes (rajiv, cricket)

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

(*d*) red (roses)

(*e*) father (john, bob).

EXAMPLE 3: Given formulas S_1 and S_2 below. Show that $Q(a)$ is a logical consequence of the two –

$S_1 : (\forall x)\,(P(x) \rightarrow Q(x))$

$S_2 : P(a)$

SOLUTION: As $(\forall x)\,(P(x \rightarrow Q(x))$

Now, $x = a$ in S_2

i.e., $Q(a)$ is a logical consequence of the two statements S_1 and S_2.

EXAMPLE 4: Translate the following statements in FOPL—

(*i*) All empolyees earning more than Rs. 60,000 per year pay income tax.

(*ii*) No employee of University earns more than the VC.

(*iii*) Only old people get sick.

(*iv*) All men are people.

(*v*) All Pompeians were Romans.

(*vi*) All Romans were either loyal to Caesar or hated him.

(*vii*) People only try to assassinate rulers they are not loyal to.

(*viii*) Everyone is loyal to someone.

(*ix*) It is now 2013.

(*x*) Mohan has exactly two friends.

(*xi*) American vegetarians do not take milk but Indian vegetarians do.

(*xii*) For every natural number, there is a natural number greater than it.

(*xiii*) Nothing beautiful is evil.

(*xiv*) Some highly qualified scientists are unemployed in Russia but in America all qualified scientists are employed.

SOLUTION: (*a*) For all x, if x is an employee and x earns more than 60,000, x pay tax.
Let employee be EMPLOYEE and pay tax be TAX.
Then, we get—
$\forall x$: ((EMPLOYEE (x) & GE ($i(x)$, 60,000) $\rightarrow$ TAX(x)).

(*b*) Let employees = x
and $VC = y$
For all x and for all y, if x is employee and y is VC then x cannot earn more than y.

$\therefore$ $\forall xy\,((E(x) \,\&\, P(y) \rightarrow \sim GE(i(x), i(y))$.

(*c*) There exists some x, if x is people and x is old,
Let people be $P(x)$,
Old be $O(x)$,
Sick be $S(x)$

$\therefore$ We get—

$\exists x$: $(P(x) \,\&\, O(x) \rightarrow S(x))$

(*d*) For all x, if x is men, x is people.

Let Men $\rightarrow$ MEN

People $\rightarrow$ PEOPLE

$\therefore$ $\forall x$: MEN $(x) \rightarrow$ PEOPLE(x)

(*e*) For all x, if x is Pompeians, x were Romans.

$\forall x$: Pompeian $(x) \rightarrow$ Roman (x)

(*f*) For all x, if x were Romans, x were loyal to Caesar or x hated him.

$\therefore$ $\forall x$: Roman $(x) \rightarrow$ LOYAL $(x,$ Caesar$) \vee$ HATE $(x,$ Caesar$)$

(*g*) $\forall xy$: PEOPLE$(x) \wedge$ RULER$(y) \wedge$ ASSASSINATE $(x, y) \rightarrow \sim$ LOYAL, To (x, y)

(*h*) For all x, x is loyal to y.

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

(*i*) now = 2013

(*j*) Let $E(x)$ where x is a person,

$M(x)$ where x is a Mohan.

$F(x)$ x has friend.

a for 2 as constant

$\therefore$ $\forall x\,(M(x) \rightarrow F(a, x))$

(*k*) Let $E(x) - x$ is vegetarian.

$A(x) - x$ is American.

$I(x) - x$ is Indian.

$M(x) - x$ takes milk.

$\sim M(x) - x$ do not take milk.

$\therefore$ $\exists x\,(E(x)\ \&\ A(x) \rightarrow \sim M(x))$

$\exists x\,(E(x)\ \&\ I(x) \rightarrow M(x))$

(*l*) For all x and y, if x is natural number and y is natural number, y is greater than x.

Let $N(x)$ be x as natural number,

$N(y)$, y is a natural number.

$G(x, y)$, x is greater than y.

$\forall x, y\,(N(x)\ \&\ N(y) \rightarrow a\,(y, x))$

(*m*) For every x, if x is nothing beautiful, x is evil.

$\forall x\,(\sim B(x) \rightarrow E(x))$.

This can also be written as—

For every x, if x is nothing and x is not beautiful, x is evil.

$\forall x : (\sim B(x) \rightarrow N(x) \rightarrow E(x))$

where $B(x)$, x is beautiful. $N(x)$, x is nothing.

$E(x)$, x is evil.

(*n*) Let us denote the following—

$S(x)$: x is highly qualified scientist

$A(x)$: x is American (scientist in America)

$R(x)$: x is Russian (scientist in Russia)

$E(x)$: x is employed.

$\sim E(x)$: x is not employed

$\therefore \quad \forall x : (S(x) \,\&\, A(x) \rightarrow E(x))$

$\exists x \,(S(x) \,\&\, R(x) \rightarrow E(x))$

EXAMPLE 5: Convert into FOPL statements:

(*i*) If x is connected to y by z, y is connected to x by z.

(*ii*) If you can get to y from z and you can get to z from y, you can get to z from x.

(*iii*) If x is connected to town y by highway z and bicycles are allowed on z, you can get to y by bike from x.

(*iv*) Town A is connected to Town B by Road 1.

(*v*) Apples are food.

(*vi*) Bill eats peanuts and is still alive.

(*vii*) Bill is brother of SUE.

(*viii*) All Pompeians died when volcano erupted in 79 A.D.

(*ix*) All men are mortal.

(*x*) Town–A and Town–E are not connected by Road 3.

(*xi*) Bikes are allowed on road-3, road-4 and road-5.

SOLUTION: (*i*) $\forall x, y, z$: (CONNECTED $(x, y, z) \rightarrow$ CONNECTED (y, x, z))

(*ii*) $\forall x, y, z$: (GETTO (z, y) and GETTO $(y, z) \rightarrow$ GETTO (x, z))

(*iii*) $\forall x, y, z$ (CONNECTED (x, y, z) and BIKE $(z) \rightarrow$ GETTO (x, y))

(*iv*) $\forall a, b$ road-1 CONNECTED $(a, b,$ road 1)

(*v*) For all x, if x is apple, x is food.

$A(x)$: x is apple.

$F(x)$: x is food.

$\therefore \quad \forall x : (A(x) \rightarrow F(x))$

(*vi*) $B(x)$: x is bill.

$P(x)$: x eats peanuts.

$A(x)$: x is alive.

$\therefore \quad \forall x : (B(x) \rightarrow P(x) \,\&\, A(x))$

(*vii*) Brother (Bill, Sue)

(*viii*) Erupted (volcano, 79) $\wedge \forall x$ [Pompeian $(x) \rightarrow$ died $(x, 79)$]

(*ix*) For all x if x is men, x is mortal.

(*x*) If x is town A and y is town B, x and y are not connected by road-3.

$\therefore \quad \forall x, y, z : \sim$ CONNECTED (x, y, z) or

$\forall x, y, z$: (Town (x) & Town $(y) \rightarrow \sim$ CONNECTED (x, y, z)

(*xi*) $\forall x$: BIKE (x)

$\forall y$: BIKE (y)

$\forall z$: BIKE (z).

Inferencing in predicate logic

The rules of inference for the propositional formulas are applicable to predicate calculus. So propositional formulas are also predicate formulas. We have to just replace propositional variables by predicate variables.

For example.

1. $\exists x(P(x) \vee Q(x)) \equiv \exists x\, P(x) \vee \exists x\, Q(x)$
2. $\exists x(P \vee Q(x)) \equiv P \vee (\exists x\, Q(x))$
3. $\forall x(P(x) \wedge Q(x)) \equiv \forall x\, P(x) \wedge \forall x\, Q(x)$
4. $\forall x\,(P \wedge Q(x)) \equiv P \wedge (\forall x\, Q\,(x))$
5. $\neg(\exists x\, P(x)) \equiv \forall x \neg (p\,(x))$
6. $\forall x\, P(x) \Rightarrow \exists x\, P(x)$
7. $\neg(\exists x\, P(x)) \equiv \exists x \neg P(x))$

We are in a position to solve some problems now.

Q. 1. Represent the following in symbol logic:

(*a*) All that glitters is not gold.

(*b*) Any person who is respected by every person is a king.

(*c*) God helps those who help themselves.

(*d*) Jack and Jill went up the hill.

(*e*) Ram likes Sita.

Ans. (*a*) $\neg \forall x$ Glitters $(x) \rightarrow$ Gold

(*b*) $\exists x \forall y$ King (x)

(*c*) $\forall x$ helps (God, helps (x, x))

(*d*) together (went up (jack), went up (jill) $\rightarrow$ hill

(*e*) likes (Ram, Sita)

Resolution in Predicate Logic

In predicate logic, we need to compare the arguments of the literals also. In predicate logic, the existential and universal quantifiers are used. We do **skolemisation for that skolemisation is removing existential quantifiers and replacing the corresponding variable by a constant or a function.** A constant or a function is called as **skolem constant or function** respectively.

Resolution Steps

It requires that all statements be converted into a nonmalized clausal form. Its procedure is given below:

Step 1: Eliminate all implication and equivalency connectives (use $\sim P \vee Q$ in place of $P \rightarrow Q$ and $(\sim P \vee Q)$ and $(\sim Q \vee P)$ in place of $(P \leftrightarrow Q)$

Step 2: Move all negations in, to the immediately preceding atom. Use P in place of $\sim(\sim P)$ and De Morgans's Law, $\exists x \sim F[x]$ in place of $\sim(\forall x)F(x)$ and $\forall x \sim F[x]$ in place of $\sim(\exists x)(F[x])$.

Step 3: Rename Variables, if necessary, so that all quantifiers have different variable assignments, i.e., rename variables so that variables bound by one quantifier are not the same as variables bound by a different quantifier.

For example.: In the expression $\forall x(P(x)) \rightarrow (\exists x(Q(x)))$

rename the second "**dummy**" variable x which is bound by the existential quantifier to be a different variable, say y, to give $\forall x(P(x) \rightarrow (\exists y\, Q(y)))$.

Step 4: **The process of eliminating the existential quantifiers through a substitution process requires that all such variables be replaced by something called "Skolem functions" which are the arbitrary functions which can always assume a correct value required for an existentially quantified variable. In this 4th step we skolemize by replacing all existentially quantified variables with skolem functions.**

Step 5: Move all universal quantifiers to the left of the expression and put the expression on the right into CNF.

Step 6: Eliminate all universal quantifiers and conjunctions since they are retained implicitly. The resulting expressions are 'clauses' and the set of such expressions is said to be in clausal form.

EXAMPLE. Convert the expression: $\exists x\, \forall y\, (\forall z\, P(f(x), y, z)$

into clausal form: $\rightarrow (\exists u\, Q(x, u)$ and $\exists v\, R(y, v)))$

SOLUTION.

Step 1: Applying step (1):

$$\exists x\, \forall y\, (\sim (\forall z)\, P(f(x), y, z) \vee (\exists u\, Q(x, u) \text{ and } (\exists v)\, R(y, v)))\ (\exists v)\, R(y, v)))$$

$[\because\ F \rightarrow G = \sim F \vee G]$

Step 2: Applying step (2) we get —

$$\exists x \forall y\, (\exists z \sim P(f(x), y, z) \vee (\exists u\, Q(x, u) \text{ and } (\exists v)\, R(y, v)))\ (\exists v)\, R(y, v)))$$

Step 3: Step (3) is not required.

Step 4: $\forall y\, (\sim P(f(a), y, g(y)) \vee (Q(a, h(y)) \text{ and } R(y, I(y)))$

[$\because$ u exists as an existential quantifier, x **exists as an existential quantifier and so does** v. So, in this step, we need to remove them and this can be done by replacing x with a, u with $h(y)$ and v with $I(x)$.]

Step 5: After applying step 5, the result is

$$\underbrace{\forall y\, ((\sim P(f(a), y, g(y)) \vee Q(a, h(y))}_{F_1} \text{ and } (\sim P(f(a), y\, g(y)) \vee R(y, I(y)))$$

Step 6: After application of step 6 we obtain:

$$\sim P(f(a), y, g(y)) \vee Q(a, h(y))$$

$$\sim P(f(a), y, g(y) \vee R(y, I(y))$$

Thus, given FOPL sentence has been converted into clausal form.

Inference Rules in FOPL

Like *PL*, a key inference rule in FOPL is modus porens. *For example.*:

Assertion: *LION* (*leo*)

Implication: $\forall x$ *LION* $(x) \rightarrow$ *FE ROCIOUS* (x)

Conclusion: *Ferocious* (*leo*)

Representation using rules

Rules can be considered as a subset of predicate logic. They have become a popular representation scheme for expert systems (also called as rule-based systems). They were first used in general problem solver (GPS) system in early (1970s). **Rules have two component parts: a LHS referred**

to as the antecedent, premise, condition or situation and a RHS known as the consequent conclusion, action or response. LHS is also known as the if part and the RHS as the then part of the rule. Some rules also include an also part. Examples of such rules are:

IF : The temperature is greater than 95°C.

THEN: Open the relief valve.

These rules are stored in KB. The interpreter or inference engine inspects the LHS of each rule in KB and replaces the contents of working memory by the RHS of the rule. **We are in a position to solve some examples now.**

Q. 1. Resolve the following example wff into its clausal form.

$$(\forall x)\{P(x) \rightarrow \{(\forall y)[P(y) \rightarrow P(f(x, y))] \wedge \sim (\forall y)[Q(x, y) \rightarrow p(y)]\}\}$$

Ans. S1: Eliminate implication symbols — all occurrences of the $\rightarrow$ symbol in a wff are eliminated by making the substitution $\sim X1 \vee X2$ for $X1 \rightarrow X2$ throughout the wff. So, this substitution yields:

$$(\forall x)\{\sim P(x) \vee \{(\forall y)[\sim P(y) \vee P(f(x, y))] \wedge \sim (\forall y)[\sim Q(x, y) \vee p(y)]\}\}$$

S2: Reduce Scopes of negation symbols — We want each negation symbol, ~, to apply to at most one atomic formula. By making use of de Morgan's laws we get:

$$(\forall x)\{\sim P(x) \vee \{(\forall y)[\sim P(y) \vee P(f(x, y))] \wedge (\forall y)[Q(x, y) \wedge \sim p(y)]\}\}$$

S3: Standardize variables—Instead of writing, $(\forall x)[P(x) \rightarrow (\exists x)\, Q(x)]$, we write

$$(\forall x)[P(x) \rightarrow (\exists y)\, Q(y)]$$

i.e., one type of dummy variable should be used with each type of quantifier. So in our wff, x-variable (dummy viz.) is used with universal ($\forall$) quantifier whereas y is used with Existential quantifier ($\exists$) . Standardizing our example wff yields:

$$(\forall x)\{\sim P(x) \vee \{(\forall y)[\sim P(y) \vee P(f(x, y))] \wedge (\exists w)[Q(x, w) \wedge \sim P(w)]\}\}$$

S4: Eliminate existential quantifiers—Consider the wff: $(\forall y)[(\exists x)\, P(x, y)]$ which might be read as "for all y, there exists an x (possibly depending on y) such that $P(x, y)$." Note that because existential quantifier is within the scope of a universal quantifier, we allow the possiblity that value x that exists might depend on the value of y. Let this dependence be explicitly defined by some function $P(x, y)$. which maps each value of y into x that "exists". Such a function is called a **skolem function**. If we use the skolem function in place of the x that exists, we can eliminate the existential quantifier 'all' together and write $(\forall y)\, P[g(y), y]$.

> NOTE. If the existential quantifier being eliminated is not within the scope of any universal quantifiers, we use a skolem function of no arguments, which is just a constant. Thus, $(\exists x)\, P(x)$ becomes $P(A)$ where the constant symbol A is used to refer to the entity that we know exists.

Eliminating all existentially quantified variables from given (there is just one) in our wff yields:

$$(\forall x)\{\sim P(x) \vee \{(\forall y)[\sim P(y) \vee P(f(x, y))] \vee [Q(x, g(x)) \wedge \sim P(g(x))]\}\}$$

where $\boxed{w = g(x)}$; $g(x)$ is a skolem function.

S5: Convert to prenex form—We may now move all of the universal quantifiers to the front of the wff. The resulting wff is said to be in **prenex form**. So we get:

$$(\forall x)\,(\forall y)\{\sim P(x) \vee \{[\sim P(y) \vee P(f(x, y))] \wedge [Q(x, g(a)) \wedge \sim P(g(x))]\}$$

S6: Put the matrix in CNF—We may put any matrix into CNF by repeatedly using one of the distributive rules, i.e.,

$$\times 1 \vee (\times 2 \wedge \times 3) \text{ by } (\times 1 \vee \times 2) \wedge (\times 1 \vee \times 3).$$

So, our equation wff becomes:

$$(\forall x)(\forall y)\{[\sim P(x) \vee \sim P(y) \vee P(f(x, y))] \wedge [\sim P(x) \vee Q(x, g(x))] \wedge [\sim P(x) \vee p(g(x))]\}$$

S7: Eliminate universal quantifiers—We get:

$$\sim P(x) \vee \sim P(y) \vee P(f(x, y)) \wedge [\sim p(x) \vee Q(x, g(x))] \wedge [\sim P(x) \vee \sim P(g(x))]$$

S8: Eliminate $\wedge$ symbols—We get

$$\left.\begin{array}{l} \sim P(x) \vee \sim P(y) \vee P[f(x, y)], \\ \sim P(x) \vee Q[x, g(x)], \\ \sim P(x) \vee \sim P[g(x)] \end{array}\right\} \text{ is required clausal form}$$

Q. 2. Show the validity of the following sentences."All men are mortal. John is a man. Therefore John is mortal."

$$(\forall x)\{P(x) \rightarrow \{(\forall y)[P(y) \rightarrow P(f(x, y))] \wedge \sim (\forall y)[Q(x, y) \rightarrow p(y)]\}\}$$

Ans. Say,

MAN (x)— x is a man.

MORTAL (x)— x is mortal.

In *FOPL*, we have

$$((\forall x)(\text{MAN}(x) \rightarrow \text{MORTAL}(x)) \wedge \text{MAN (John)}) \rightarrow \text{MORTAL (John)}$$

Convert it to Skolem form —

$$\propto : \sim((\text{MAN}(x) \vee \text{MORTAL}(x)) \wedge \text{MAN (John)}) \vee \text{MORTAL (John)}$$

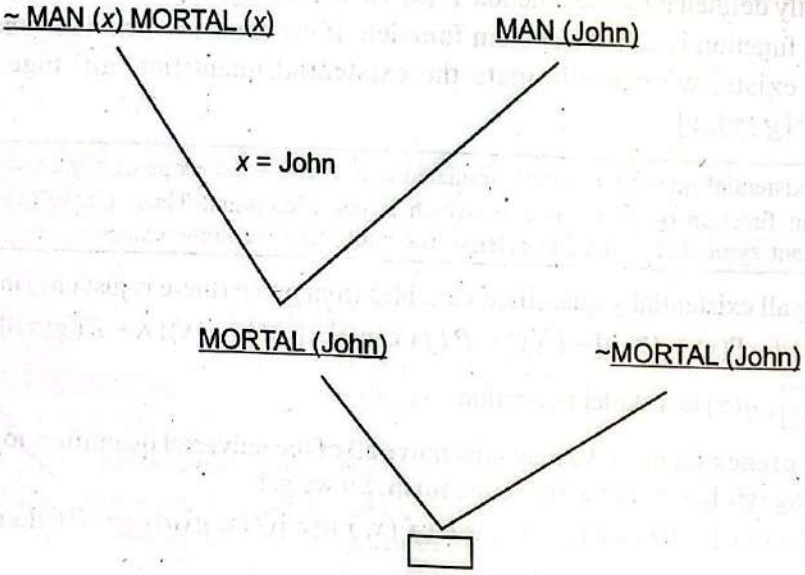
If we show that $\sim\propto$ is unsatisfiable then $\propto$ is valid.

$$\sim \propto : (\sim \text{MAN}(x) \vee \text{MORTAL}(x)) \wedge \text{MAN (John)} \wedge \sim \text{MORTAL (John)}$$

$\therefore$ A set of clauses, S, that can be obtained from $\sim\propto$ is —

$$S = \{\sim \text{MAN}(x) \vee \text{MORTAL}(x), \text{MAN (John)}, \sim \text{MORTAL (John)}\}$$

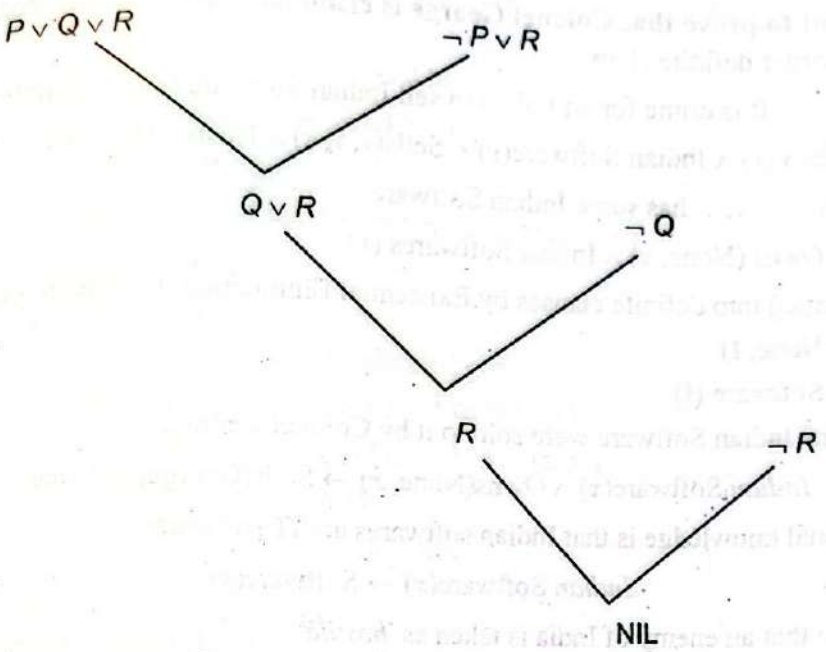
If we show that S is unsatisfiable then $\sim\propto$ is also unsatisfiable and hence $\propto$ is valid. In tree form,



Q. 3. Given the following predicates, show how resolution process can be applied to resolve —

$T: P \vee Q \vee R \quad U: \sim P \vee R \quad V: \sim Q \quad W: \sim R$

Ans. The resolution tree is —



Q. 4. Consider the following axioms —

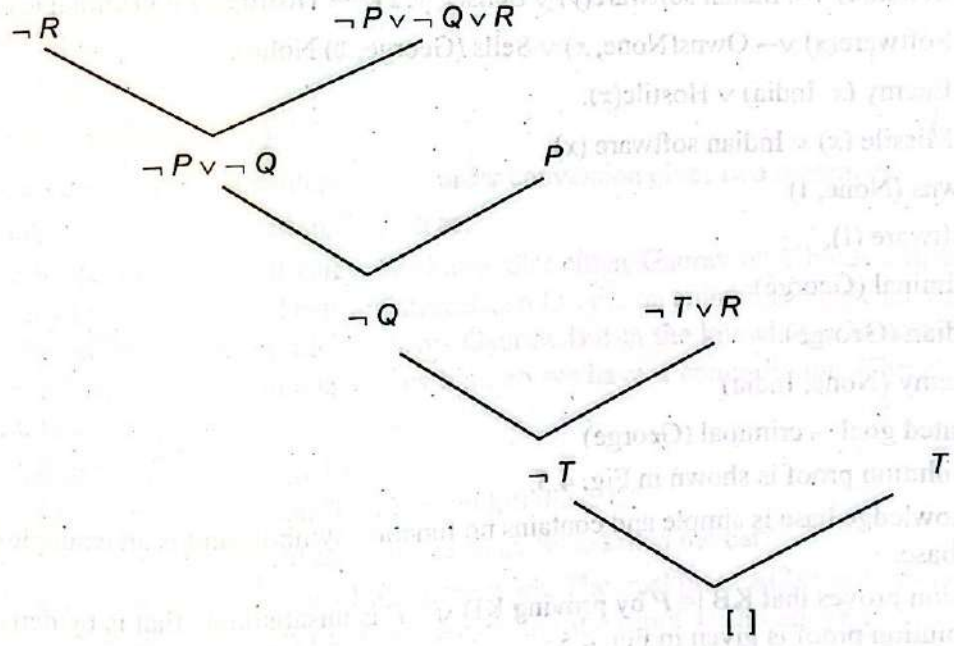
$P, (P \wedge Q) \rightarrow R, (S \vee T) \rightarrow Q, T$

Prove that R is true by resolution.

Ans. We negate the goal, i.e., we negate R, i.e, $\neg R$. We apply resolution now —

We convert the given axioms into clausal form —

1. P 2. $\neg P \vee \neg Q \vee R$ 3. $\neg S \vee Q$ 4. $\neg T \vee Q$ 5. T



Q. 5. Consider the following problem:

The law says that it is crime for an Indian to sell Indian Software to hostile nations. The country, named None, an enemy of India has some Indian software and all its software were sold to it by Colonel George who is an Indian. Give its resolution proof.

Ans. We want to prove that Colonel George is criminal. First, we shall represent these facts as first order definite clauses:

1. .....It is crime for an Indian to sell Indian Software to hostile nations.

   Indian (x) $\wedge$ Indian Software(y) $\wedge$ Sells(x, y, z) $\wedge$ Hostile (z) $\rightarrow$ Criminal (x)

2. None ..... has some Indian Software.

   $\exists x$ *Owns* (None, x) $\wedge$ Indian Softwares (x)

which is transformed into definite clauses by Existential Elimination, introducing new constant I

Owns (None, I)

Indian Software (I)

3. All of its Indian Software were sold to it by Colonel George.

   Indian Software(x) $\wedge$ Owns(None, x) $\rightarrow$ Sells(George, x, None)

   Additional knowledge is that Indian softwares are IT softwares

   Indian Software(x) $\rightarrow$ Software(x)

   and also that an enemy of India is taken as '*hostile*'

   Enemy(x, India) $\rightarrow$ Hostile(x)

4. *George who is an Indian* .....

   Indian (George)

5. The country None, an enemy of India .....

   Enemy (None, India)

In CNF they are:

(*a*) $\neg$ Indian(x) $\vee \neg$ Indian software(y) $\vee$ Sells(x, y, z) $\vee \neg$ Hostile(z) $\vee$ criminal (x).

(*b*) $\neg$ Software(x) $\vee \neg$ Owns(None, x) $\vee$ Sells (George, x) None).

(*c*) $\neg$ Enemy (x, India) $\vee$ Hostile(x).

(*d*) $\neg$ Missile (x) $\vee$ Indian software (x).

(*e*) Owns (None, I).

(*f*) Software (I).

(*g*) Criminal (George)

(*h*) Indian (George)

(*i*) Enemy (None, India)

and the negated goal $\neg$ criminal (George)

The resolution proof is shown in Fig. 4.5.

This knowledge base is simple and contains no function symbols and is an example of Datalog knowledge base.

Resolution proves that KB $\models P$ by proving KB $\vee \neg P$ is unsatisfiable that is by deriving empty clause. Resolution proof is given in Fig. 4.5.

We may notice the "single spine" beginning with the goal clause, resolving against clauses from the knowledge base until the empty clause is generated. This is characteristic of resolution on Horn clause knowledge bases.

This example makes use of Skolemisation and involves clauses which are not definite clauses. This results in a somewhat complex structure. The problem in English is as follows:

To prove Vibhuti Killed Lucy

1. Everyone who loves all animals is loved by someone.
2. Everyone who kills an animal is loved by no one.
3. Gaurav loves all animals.
4. Either Gaurav or Vibhuti killed the cat who is named Lucy.
5. Did Vibhuti kill the cat?

Goal, G to prove that Vibhuti killed Lucy.

Conversion is First-order Logic and the negated goal.

(*a*) $\forall x\ [\forall y\ \text{Animal}(y) \rightarrow \text{Loves}\ (x,y)] \rightarrow [\exists y\ \text{Loves}\ (y, x)]$.

(*b*) $\forall x\ [\exists y\ \text{Animal}\ (y) \wedge \text{Kills}\ (x, y)] \rightarrow [\forall z \neg\ \text{Loves}\ (z, x)]$.

(*c*) $\forall x\ \text{Animal}\ (y) \rightarrow \text{Loves}\ (\text{Gaurav}, x)$.

(*d*) $\text{Kills}\ (\text{Gaurav}, \text{Lucy}) \vee \text{Kills}\ (\text{Vibhuti}, \text{Lucy})$

(*e*) Cat(Lucy)

(*f*) $\forall x\ \text{Cat}(x) \rightarrow \text{Animal}\ (x)$.

(*g*) Kills (Vibhuti, Lucy)

and conversion to CNF to each sentence gives

(*a*) $\text{Animal}\ (F(x)) \vee \text{Loves}\ (G(x), x)$

(*b*) $\neg\ \text{Loves}\ (x, F(x)) \vee \text{Loves}\ (G(x), x)$

(*c*) $\neg\ \text{Animal}\ (y) \vee \neg\ \text{Kills}(x, y) \vee \neg\ \text{Loves}(z, x)$

(*d*) $\neg\ \text{Animal}\ (x) \vee \neg\ \text{Loves}\ (\text{Gaurav}, x)$

(*e*) $\text{Kills}(\text{Gaurav}, \text{Lucy}) \vee \text{Kills}\ (\text{Vibhuti}, \text{Lucy})$

(*f*) Cat (Lucy)

(*g*) $\neg\ \text{Cat(x)} \vee \text{Animal}(x)$

(*h*) $\neg\ \text{Kills}\ (\text{Vibhuti}, \text{Lucy})$

After conversion the first sentence from order conversion gives two sentences.

The proof can be paraphrased in English as:

Suppose Vibhuti did not kill Lucy. We know that either Gaurav or Vibhuti did; thus Gaurav must have. Now Lucy is a cat and cats are animals, so Lucy is an animal. Because anyone who kills an animal is loved by no one, so no one loves Gaurav. But in the knowledge base it is given that Gaurav loves all animals, so someone loves him, so we have a contradiction. Therefore, Vibhuti killed the cat, Lucy.

The Resolution is given in Fig. 4.6.

The proof answers the question "Did Vibhuti kill the cat?"

But often we ask more general question, such as "who killed the cat?"

Resolution can do this by doing a little more work. The goal is $\exists$w kills (w, Lucy) which when negated becomes $\neg$ kills (w, Lucy) in CNF. Repeating the proof in the above Fig., with the new negated goal, we obtain a similar proof tree, the substitution then becomes $w/_{\text{vibhuti}}$ in one of the

Fig 4.5. A resolution proof of "George is a criminal.

steps. (The reader should repeat the above proof with the new unification), concluding that Vibhuti is killer of cat Lucy.

Unfortunately resolution can produce non-constructive proofs for existential goals. *For example* ¬ Kills (w, Lucy) resolves with Kills (Gaurav, Lucy) ∨ Kills (Vibhuti, Lucy) to give Kills (Gaurav, Lucy), which resolves again with ¬ Kills(w, Lucy) to yield the empty clause. Here, w has two different bindings in this proof – Vibhuti or Gaurav. This is surprising, the procedure of resolution tells that someone, either Vibhuti or Gaurav, killed Lucy. **This type of double bindings is not quite common**, and can be avoided by

(*i*) Either restricting the allowed resolution steps so that w (query variable) is bound once in a given proof. Then we need to be able to backtrack over the possible bindings.

(*ii*) Or be adding a special **answer literal** to the negated goal. This then becomes

¬ Kills (w, Vibhuti) ∨ Answer(w).

At present the resolution process generates an answer only when a clause is generated containing just a *single* answer literal (answer(Vibhuti) in the Fig. 4.6. The non-constrictive proof would produce the clause

Answer (Vibhuti) ∨ Answer(Gaurav)

which is, in fact, the no answer.

Thus, we have seen how the process of deduction can derive new facts (answer) from the given facts (knowledge base). Further, resolution theorem can be used to answer yes-no questions such as Did Vibhuti kill the cat?

Resolution can also be used to answer fill-in-the-blanks type of questions, such as "Was the cat killed?" or " Who killed the cat?" Even "When was the cat killed" can be answered by adding new facts in the knowledge base. But yes-no and fill-in-the-blank questions are not the only kinds of questions which can ever be asked.

This must be mentioned here that the technique of deduction does not contain the answer to all conceivable questions. In general, it is necessary to decide on the kinds of questions which will be asked and to decide a *knowledge representation* method appropriate for those questions. *For example,* even we might ask "How was the cat killed?" or "Why was the cat killed?" So method of resolution is not yet complete and needs further research.

4.3 UNIFICATION ALGORITHM

Unification is the recursive process of merging two or more predicates with constants or with some variables. We check whether by performing some substitution for variables, the predicates can be made identical.

For example. Small town (*x*)¬ Small town (Delhi)

Cannot be merged (or resolved) because there may be various values for *x*.

Similarly, predicate lady and lady (Ajay) cannot be resolved as the first predicate doesn't have any variable and second predicate has some variable / constant.

Unification is used for unifying the axioms in inferencing mechanism. It is a recursive process. The substitutions for the variables are tried to generate identical and opposite predicate sets such that they can be combined. Unification tries to find out the complementary literals in two clauses and deletes them, thereby forming a new literal. **Please understand that whenever two identical and opposite literals are obtained, they are merged (or resolved).** This is true for clauses having no variables. When the literals have variables, the process becomes complicated. Thus, we use **substitutions** here. Substitution may be of a variable by a constant, of a variable by another variable or of a variable by a function that doesn't contain the same variable.

Fig 4.6. A resolution proof of "Vibhuti killed the cat.

Let us write the unification algorithm first.

Algorithm: Verify: (U, V)

1. If U and V are both variables or constants then
   (*a*) If U and V are identical, then return null.
   (*b*) If U is a variable then if U occurs in V, then return {*FAIL*} else return {U/V}.
   (*c*) If V is a variable, then if V occurs in U, return {FAIL} else return {U/V}.
   (*d*) Return {FAIL}
2. If the initial predicate symbols in U and V are not identical, return {FAIL}.
3. If U and V have a different number of arguments, then return {FAIL}.
4. Set SUBSET to NULL.
5. For $i \leftarrow 1$ to the number of arguments of U.
   (*a*) Call unify with the i^{th} argument of U and the i^{th} argument of V putting the result in S.
   (*b*) If S contain {FAIL}, return {FAIL}?
   (*c*) If S is not equal to NULL
      (*i*) Apply 5 to be the remainder of both U and V.
      (*ii*) Set subset equal to APPENDS (S, SUBSET).
6. Return SUBSET.

Working: We try to unify, say, the following predicates —

$P(x, x)$

and $\neg P(y, z)$

Firstly, we check the predicates. Here these are same and opposite. Next, we check their arguments. Now, P has its 1st argument as 'x' whereas $\neg P$ has 'y' as its 1st argument./ These can be made similar if we substitute x in place of y in both predicates. **Please note here that since x and y are variables so these substitutions are possible.** Many a time, the situation is that one predicate has variable and another predicate has constant, then only variable can be replaced by the constant. the constant cannot be replaced by variable. **Also note that the replacement of one variable by another should be done throughout all occurrence of that variable in all predicates.** The replacement of y by x is denoted by x/y. So, now the predicates are —

$P(x, x)$

and $P(x, z)$

Now, we come to the 2nd argument of both the predicates. To match, x should be replaced by z. But this cannot be done as the substitution of x by y and z both is not possible in one set of problem. Hence, the two literals cannot be unified.

On the other hand, if predicates are —

$P(x, x)$

and $\neg P(y, y)$

And if we substitute x/y (i.e., x for y) then they become

$\neg P(x, x)$

and $\neg P(x, x)$ and these can be **unified.**

We are in a position to solve some examples now.

Q.1. Unify the following clauses —

$P(x, f(z), \text{mary})$

$\neg P(x, y, z)$.

Ans. We substitute z/mary in both the clauses —

$P(x, f(\text{mary}), \text{mary})$

$\neg P(x, y, \text{mary})$. Now, these two clauses

differ in their 2nd argument only, so we replace $y/f(\text{mary})$ and we get —

$\neg p(x, f(\text{mary}), \text{mary})$.

$\therefore$ Finally, the two clauses are —

$P(x, f(\text{mary}), \text{mary})$

and $\neg p(x, f(\text{mary}), \text{mary})$

$\therefore$ Now, they can be unified (or merged) together.

Q. 2. Suppose we have the following query — Whom does Ravana know? And that there are four sentences in the KB.

Knows (Ranana, Sita)

Knows (y, Rama)

Knows(y, Mother (y))

Knows (x, Radha)

Unify given sentence with the above four sentences.

Ans. Given: Sentence is.

Knows (Ravana, x) ...(1)

$\therefore$ Following results are obtained by unifying (1) with given sentences

1. UNIFY (Knows (Ravana, x), Knows (Ravana, Sita)
   = (x/Sita)
2. UNIFY (Knows (Ravana, x), knows(y, Ram)
   = (x/Ram, y/Ravana)
3. UNIFY (Knows (Ravana, x), knows (y, Mother(y))
   = (y/Ravana, x/mother(Ravana))
4. UNIFY (Knows (Ravana, x), knows (x, Radha) = fail

In this last unification, the two constants cannot be instantiated to the same variable 'x' at the same time. This problem can be solved by assigning **different variables** to two sentences. 'x' in the knows (x, Radha) can be changed to 'y'. Now, the unification will work by standardising one of the two sentences being unified which means renaming its variables to avoid name clashes.

For example, renaming 'x' in knows (x, Radha) to a new variable 'y' without changing its meaning is as follows.

UNIFY (knows (Ravana, x); knows (y, Ravana)
= [x/Radha, y/Ravana]

NOTE. 1. Say, we are attempting to unify the expression

$F(x, x)$

$f(g(x), g(x))$

That is, if we accept $g(x)$ as a substitution for 'x' then we would have to substitute it for 'x' in the remainder of the expressions. But this leads to **infinite recursions** since it is not possible to eliminate x.

2 Unification works well for two arguments. But there could be more than one such **Unifier.**

For example Consider again,

UNIFY knows (Ravana, x), knows (y, z)

$\therefore$ Substitutions could be

(y/Ravana, x/z)

or (y/Ravana, x/Ravana, z/Ravana)

First substitution gives, knows (Ravana, z) and **Second substitution** gives knows (Ravana, Ravana).

The second result can be obtained from the first (y/Ravana, x/z) by an additional substitution (z/Ravana). **Please note that the first unifier is more general than the second because the second can be derived from the first.** Also we conclude that for every unifiable pair of expressions there is a single **most-general-unifier** (MGU). In this example, it is (y/Ravana, x/z).

4.4 FORWARD CHAINING, BACKWARD CHAINING, AND CONFLICT RESOLUTION

A.I. solves our problems by searching a solution search-tree (called as search space) from the initial state to the goal state. We have rule-based system architecture that has a set of rules, a set of facts and an inference engine. Given a set of rules, the new knowledge can be inferred using two rules —

(*a*) Forward chaining / Forward reasoning.

(*b*) Backward chaining / Backward reasoning.

Forward chaining (or reasoning) starts from the start state and proceeds towards the goal state. On the other hand backward chaining starts from the goal state and proceeds towards the start state.

For example. : Turbo Prolog (an A.I. language, 5 G), uses backward chaining (or reasoning).

Forward reasoning / data-driven inferencing

As we have already studied that rules are stored in KB in form of atomic formula in some form like propositional or predicate logic. **The process of instantiating the left side of rules, executing them from left to right, matching the left part of the sentence with the existing expression and if the match occurs then replacing it by the right part of the rule is called as Forward reasoning or Forward chaining or Data-driven search.** In next step, this right hand side of the previous rule is considered as left side of present rule to find applicability of a rule. So the finding of the rule becomes *Subgoals* to be fulfilled. These subgoals may in turn cause new subgoals to be established and so on, until the facts are found to match the lowest subgoals conditions. **Whenever the goal is achieved, the search process stops.**

For example. Say, we have a database with the following elements —

A	G	E
A	C	B

(database having facts)

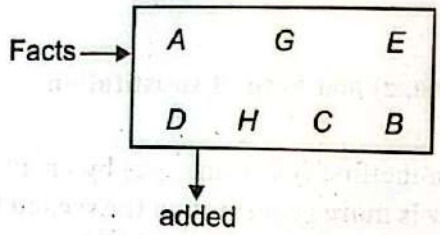
& the set of rules as —

$F \& B \rightarrow Z$
$C \& D \rightarrow F$
$A \rightarrow D$

(Rules in a *KB*)

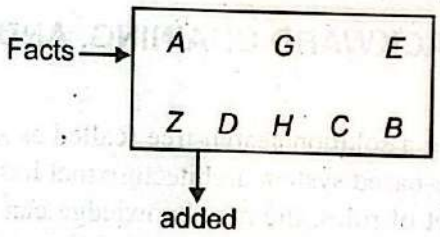
We assume that each time the set of rules is tested against the database, only the first (topmost) rule that matches is executed. So the rule $A \rightarrow D$ is executed only once although it matches the database every time. **This database is also known as a working memory (WM).**

How it works? The first rule which fires is $A \rightarrow D$ because A is already in the database. So, this rule shows the existence of D (i.e., inferring) and D is placed in the database. So, our database now is —



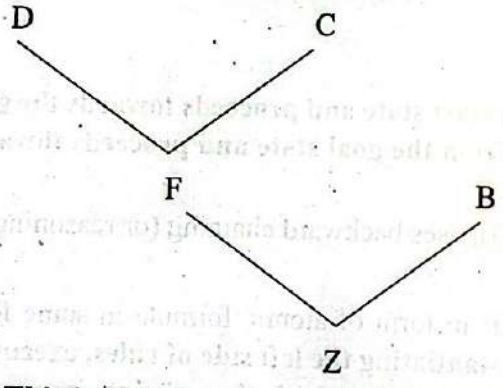
(D is also added now)

This causes the second rule C and $D \rightarrow F$ to fire and as a consequence F is inferred and placed in the database. This in turn causes the tired rule F and $B \rightarrow Z$ to fire placing Z in the database i.e.



(D is added now)

∴ The inference chain so formed is—



This is known as forward chaining. It is also called as a data-driven search as the input data is used to guide the direction of inference process.

Let us write its algorithm now.

Algorithm: Forward chaining

Repeat: Collect the rule whose condition matches a fact in database (or WM).
Do actions indicated by the rule — added facts to WM 'or delete facts from *working memory*).
Until: Problem is solved or no condition matches.

Alternative Algorithm: Forward Chaining

Repeat
Collect the rules whose conditions match facts in database (or WM)
If more than one rule matches then use conflict resolution strategy to eliminate all but one.
else
do actions indicated by the rules i.e., add facts to the database/ or delete facts from WM, until a problem is solved or no condition matches.

Backward chaining/Backward reasoning/Goal-driven inference

It is a process of reasoning in which within a rule it matches the existing state with the right hand part of the rule and if the match occurs, it is replaced by left hand side of the rule. Actually, backward chaining means reasoning from goals back to facts. The focus is on the search. The rules and facts are processed using backward chaining interpreter.

Let us write its algorithm now.

Algorithm: Backward chaining

Prove goal G:

If G is in the initial facts, it is proven. Please find a rule which can be used to conclude G and try to prove each of that rules conditions.

Which one to select?

If we apply both forward and backward algorithms then our searching becomes a bidirectional search. This results in a hybrid reasoning. Which reasoning to select depends on the nature of the problem. It depends on the following factors:

(*a*) **Number of starts and goal states.**

We proceed from a smaller set of states to a larger set of states.

(*b*) **Direction of branching factor (b)**

Branching factor is defined as the average number of nodes that are generated from a single node. **Please remember that the reasoning is performed in the direction of lower branching factor (b).**

(*c*) **Problem's requirement**

Certain applications like medical diagnosis need backward chaining as they need the justification of the solution too. **Note that it is important to choose the reasoning in the direction that corresponds to the way in which the user thinks.** Actually, we can use hybrid reasoning for such systems. Programming languages, like PROLOG, support backward chaining.

We need to check the rule form. If both left and right hand side contain assertion, then forward chaining can match assertions on the left side of a rule and add to the state description of assertion on RHS. **Please understand here that if arbitrary procedures are allowed at right in the rule then a rule will not be reversible.**

Some production languages allow only reversible rules and some do not. **Also note that when irreversible rules are used then a commitment to the direction of search must be made at the time the rules are written.**

Conflict Resolution

The set of rules that have their conditions stratified by working memory elements forms a *conflict set*. **A conflict is said to occur if more than one rules are found to fire in one cycle.** In such a

scenario, the control structure must determine which rule to fire from this conflict set of ac rules. This process of selection is called as **"conflict resolution". So, conflict resolution norm selects a single rule to fire.** There are 3 conflict resoltuion mechanisms:

(*a*) Refractory (*b*) Receiving (*c*) Specificity

Refractory: In this mechanism, a rule should not be allowed to fire more than once on the sa data. This prevents undesired loops.

Recency: Here we rank instantiation in terms of recency of elements in the premise of the rule. T **rule which has highest priority is fired first.**

Specificity: More specific rules are better because they take more of data into account. It is a known as a strict condition as it injects more know ledge in the KB, when it is fired.

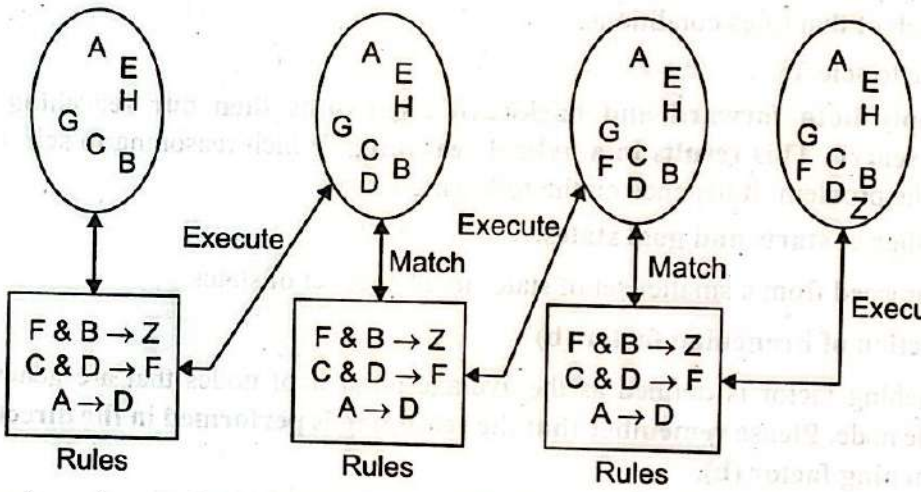
Other mechanisms are also used like—

(*a*) **Most recently used:** It chooses the most recently used rule from the conflict set for firi It represents a dfs.

(*b*) **Most recently added:** It selects the rule which is most recently added in the set of rules is a dynamic policy as in this policy new production rules are added, deleted or modif automatically in dynamic KB during execution.

We are in a position to solve an example now.

EXAMPLE 1. Consider the following database and the set of rules given below:



Explain how forward chaining works in this example. Also show backward chainin Which is more cost-effective?

SOLUTION. The rules in this example, use letters as facts in the database to explain the situatio or concepts.

For examples., F and $B \rightarrow Z$

means

It situation F exists

and situation B exists

THEN : situation Z also exists.

Let us see now how these rules work.

Assume that each time the set of rules is tested against the database, only the first (topmost rul which matches is executed. That is why, in the figure given, the rule ($A \rightarrow D$) is executed only onc even though it matches the database every time.

The first rule that fires is $A \rightarrow D$ because A is already in the database. As consequence of this rule, the existence of D is inferred and D is placed in the database. This causes the second rule, C and $D \rightarrow F$ to fire, as a consequence F is inferred and placed in the database. This in turn causes the 3rd rule F and $B \rightarrow Z$ to fire placing Z in the database. **Please understand that this technique is called as forward chaining because the search for new information seams to be proceeding in the direction of the arrow separating the left and right hand sides of the rules.** The system which uses information on the left hand side to derive information on the right is shown in Fig 4.7(a).

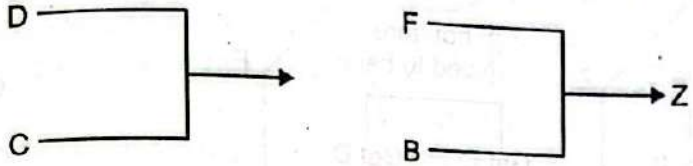


Fig. 4.7.(a) Forward chaining.

Say, we use this system with the goal of determining whether or not situation Z existed. We might think that it worked quite well, finding quickly the fact that Z did exist. Unfortunately, this is just over simplification. Practically speaking, the process of drawing inference is not simple. This is so because a **real expert system** will not have just three rules but will have hundreds or even thousands of them. If we use a system that is very large just to find out about Z many rules would be fired which had nothing to do with Z, then a large number of inference chains and situations could be derived in the process which were valid but unrelated to Z. **Please note that if our goal is to infer one particular fact Z, forward chaining could waste both time and money.** In such a case, **backward chaining** might be more cost-effective. With this inference technique, the system starts with what it wants to prove. Like, to find if the situation Z exists and as such executes those rules which are relevant to its establishment.

The Fig. 4.7(b) shows how backward chaining would work using rules given above.

Step 1: The system is told to establish (if possible) that situation Z exists. It first checks the database for Z and when that fails, searches for rules which conclude Z, i.e. have Z on the right side of the arrow. It finds the rule F and $B \rightarrow Z$ and decides that it must establish F and B in order to conclude Z.

Step 2: The system tries to establish F, first checking the database and then finding a rule which concludes F. From the rule, C and $D \rightarrow F$, the system decides that C and D must be established to conclude F.

Steps 3 to 5: In these steps, the system finds C in the database but decides it must establish A before it can conclude D. It then finds A in the database.

Steps 6 to 8: In these steps, the system executes the 3rd rule to establish D, then executes the second rule to establish F and finally executes the first rule to establish the original goal, Z. The inference chain created here is identical to the one created by forward chaining. (This is left as an exercise for the students). **But one point to remember is that the two approaches depend on the method in which data and rules are searched.**

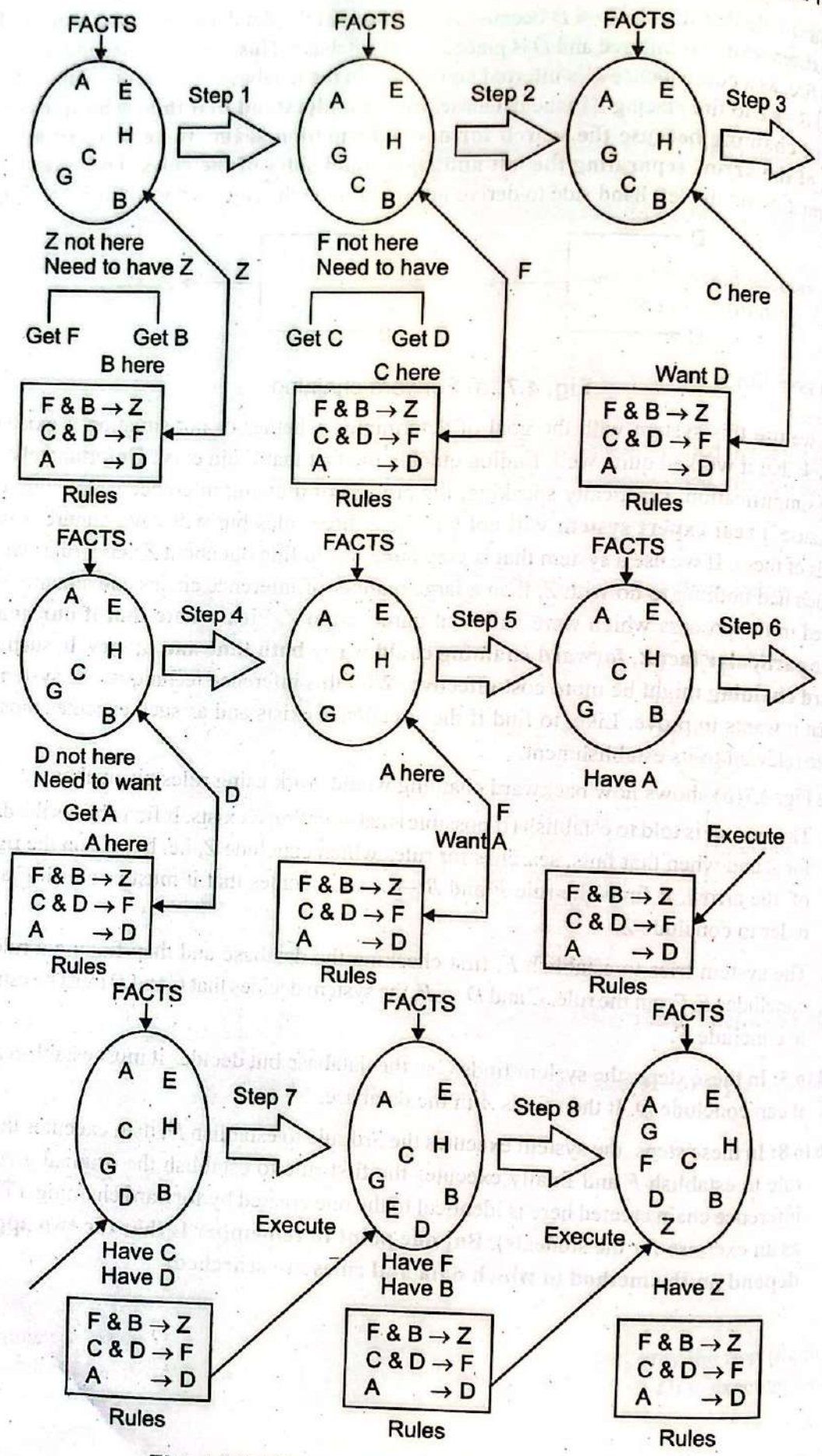


Fig. 4.7(*b*) Forward and Backward chaining.

4.5 STRUCTURED KNOWLEDGE REPRESENTATION

We have some more advanced KR techniques that focus on the structure of knowledge like semantic networks, conceptual graphs, conceptual dependencies and scripts, cyc and object oriented representation. We discuss these one by one now.

4.5.1 Slots and semantic nets/semantic network/associated networks

A slot is a subclass of intangible and is of many types. Defining slots refers not to the properties of the frame but to the properties of the object represented by the frame. **A slot is an attribute value pair in the simplest form. A filler is a value that a slot can take. For example.** it can be a numeric, string value or a pointer to another slot. A **weak slot** and filler structure does not consider the constant of the representation. They are of two types —

(*a*) Semantic networks (nets) or associated networks.

(*b*) Frames.

Let us discuss semantic networks first. They are an alternative to predicate logic as a form of knowledge representation. **The idea is that we can store our knowledge in form of a graph.** A semantic network consists of nodes connected by arcs.

Nodes represent objects in the real world.

Arcs represent relationships between those objects.

For example.

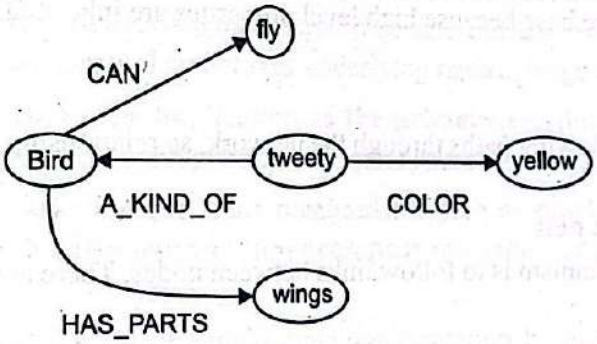


Fig. 4.8 Semantic network (net) example

Fig. 4.8 shows that tweety is a kind of bird who can fly. It is yellow in colour and has wings. These networks permit **property inheritance also. It means that nodes which are members or subsets of other nodes may inherit properties from their higher level ancestor nodes.**

For example.

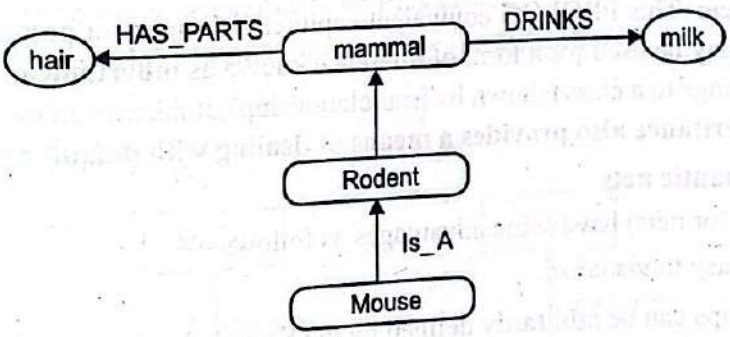


Fig. 4.9 Property inheritance in a hierarchical network

So, it is possible to infer from above net that a mouse has hair and drinks milk.

This concept of inheriting a property is called as property inheritance. The properties of a subclass can be modified and can be given different values. Each subclass will thus have its own defined values. **Please understand that the semantic network based knowledge representation mechanism is useful where an object or concept is associated with many attributes and where relationships between objects are important.**

Applications: Semantic nets have been used in natural language research to represented complex sentences expressed in English.

Properties of semantic nets

Semantic nets exhibit following properties knowledge representation scheme:

1. Expressiveness

Semantic nets allow representation of facts and relationship between facts. The levels of hierarchy provide a mechanism for representing general and specific knowledge. This representation is a model of human memory and it is therefore relatively understandable.

2. Effectiveness

They support inference through property inheritance. They can also be easily represented by *PROLOG*, *LISP* and *other A.I.* languages making them amenable to computation, as shown by "is-a" arc in LISP given by $\forall x$ [Great-Dane $(x) \rightarrow$ Dog (x)].

3. Efficiency

They reduce the size of the knowledge base. Knowledge is stored only at its highest level of abstraction rather than for every instance or example of a class. Further they help maintain the consistency in knowledge base because high level properties are inherited by subclasses and are not added for each subclass.

4. Explicitness

Reasoning equates to following paths through the network, so relationship and inference are explicit in the network lines.

Inferencing in semantic nets

The basic inference mechanism is to follow links between nodes. There are two ways of inferencing used here —

(*a*) Intersection Search. (*b*) Inheritance.

(*a*) **Intersection search: It is the process of finding the relationship among objects by spreading activation out from each of the two nodes and seeing where the activation met.** This is achieved by assigning a **special tag** to each visited node. There are many advantages including entity-based organization and fast parallel implementation. However, every structured question needs highly structured networks.

(*b*) **Inheritance:** This PROLOG equivalent captures an important property of semantic nets that they may be used for a form of inference known as **inheritance.** The idea is that if an object belongs to a class (shown by is a relationship), it inherits all of the properties of that class. **Inheritance also provides a means of dealing with default reasoning.**

Advantages of semantic nets

Semantic networks (or nets) have some advantages as follows—

1. They are easy to visualize.
2. Relationships can be arbitrarily defined by the Knowledge Engineers.
3. Formal definitions of semantic networks have been developed.
4. Related knowledge is easily clustered.

5. Are efficient in space requirements.
6. Projects are represented only once.

Disadvantages of Semantic Nets

1. Inheritance from multiple sources can cause problems.
2. Facts placed inappropriately cause problems.
3. There are no standards about node and arc value.

Types of Semantic Net

A semantic network or net is a graphic notation for representing knowledge in pattern interconnected nodes and arcs. Common to all these nets is a declarative graphic representation that can be used either to represent knowledge or to support automated systems for reasoning about knowledge. Some versions are highly informal while other versions are formally defined systems of logic. Following are the 6 most common kinds of semantic nets—

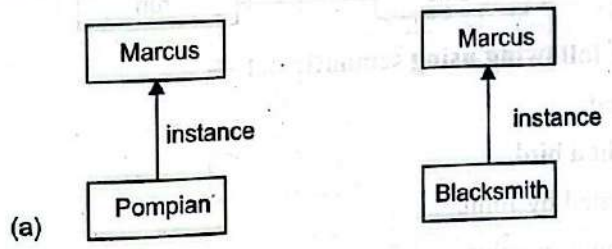
1. **Definitional networks** emphasize the subtype or is-α **relation between a concept type and a newly defined subtype.** The resulting network, also called a **generalization or subsumption hierarchy,** supports the rule of inheritance for copying properties defined for a super type to all of its subtypes. Since definitions are true by definition, the information in these networks is often assumed to be necessarily true.
2. **Assertional networks are designed to assert propositions. Unlike definitional networks 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. **Implication networks use implication as the primary relation for connecting nodes.** They may be used to represent patterns of beliefs causality, or inferences.
4. **Executable networks include some mechanism, such as marker passing or attached procedures, which can perform inferences, pass messages, or search for patterns and associations.**
5. **Learning networks** build or extend their representation 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 networks:** They combine two or more of the previous techniques, either in a single network or in separate but closely interacting networks.

We are in a position to solve some examples on nets now.

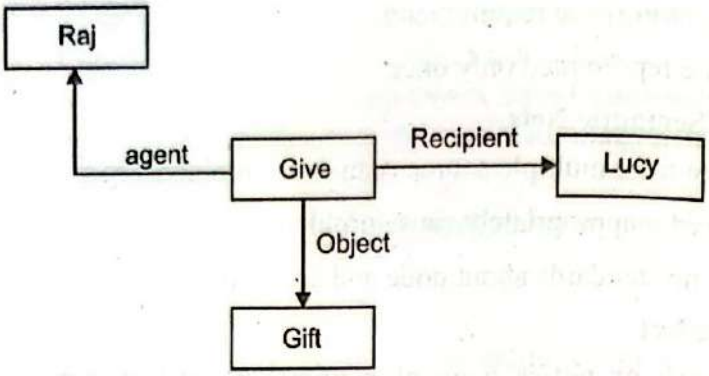
Q. 1 Construct a semantic net representation for the following—

(*a*) Pompian (Marcus), Blacksmith (Marcus).

(*b*) Raj gives Lucy a gift.



(a)

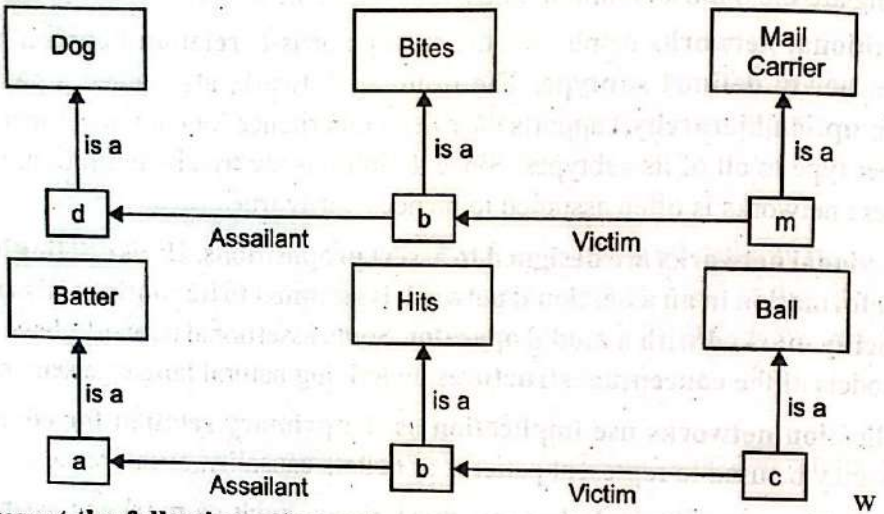


(b)

Q. 2 Construct a semantic net representation for the following —

(*a*) Every dog has bitten a mail carrier.

(*b*) Every batter hit a ball. **[MDU, BE (CSE) 8th sem. Dec. 200**



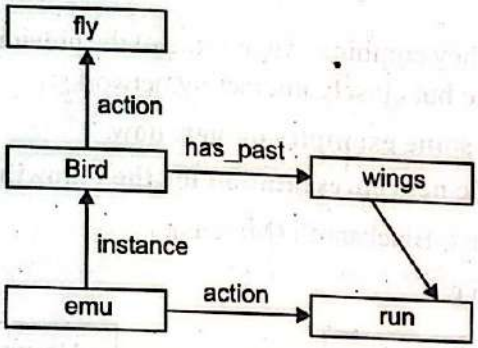
Q. 3 Represent the following using nets —

(*a*) Emus are birds.

(*b*) Typical birds fly and have wings.

(*c*) Emus run.

Ans. Its Semantic net is as follows



Q. 4. Represent the following using semantic net —

(*a*) Tom is a cat.

(*b*) Tom caught a bird.

(*c*) Tom is owned by John.

(*d*) Tom is ginger in color

(e) Cats like cream.

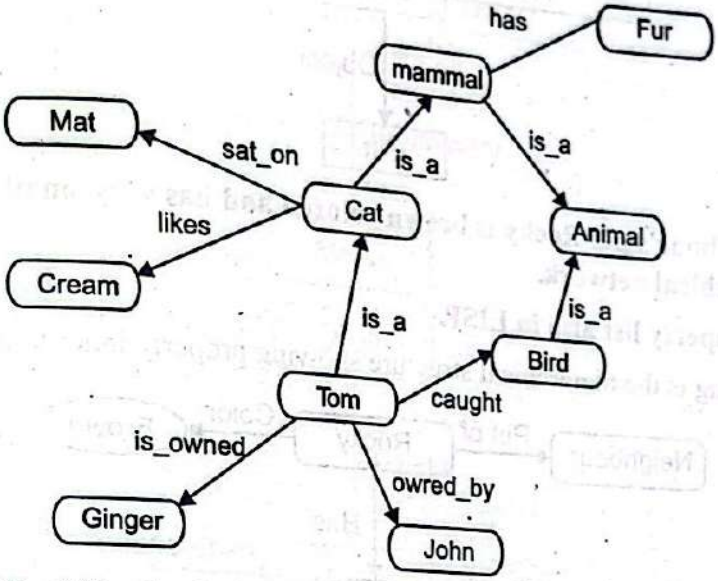
(f) The cat sat on the mat.

(g) A cat is a mammal.

(h) A bird is an animal.

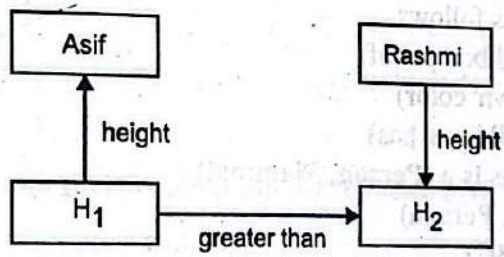
(i) All mammals are animals.

(j) Mammals have fur.



Q. 5. Represent the following in net form —

'Asif is taller than Rashmi'

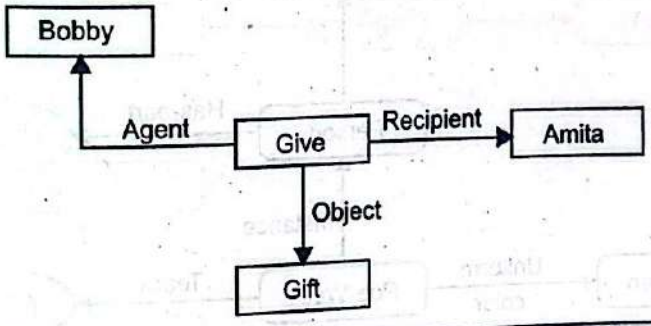


Q. 6. Draw the semantics network of the following sentences

(a) Bobby gives Amita a gift.

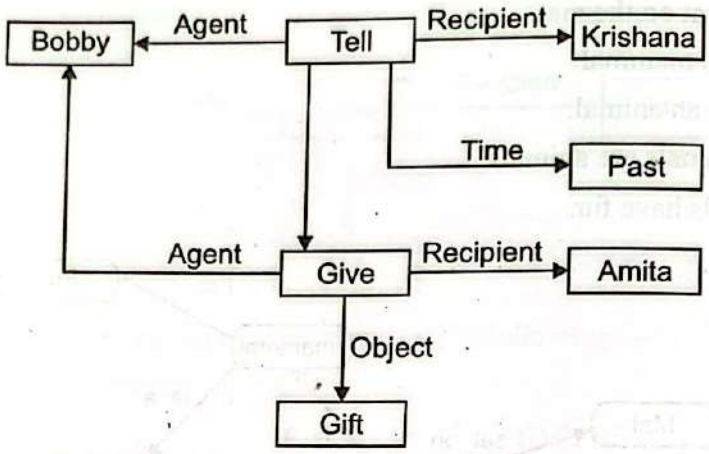
(b) Bobby told Krishana that he gave Amita a gift.

Ans. (a) Its semantic representation is as follows



NOTE: The arcs define the relationships between the predicate (GIVE) and the concepts like AMITA and GIFT, associated with that predicate.

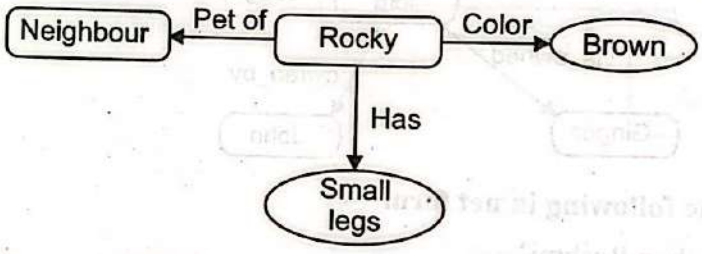
(*b*) Its semantic representation is as follows



Q. 7. "Our neighbour's pet Rocky is brown colored and has very small legs". Represent it as a hierarchical network.

Give its property list also in LISP.

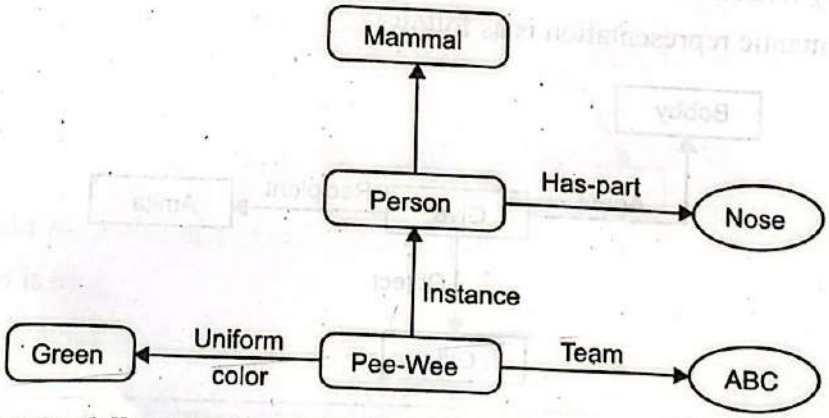
Ans. The following is the hierarchical structure showing property inheritance.



The property list is as follows.
(putprop 'rocky' neighbor' pet of)
(putprop 'rocky' brown' color)
(putprop 'rocky' small keys' has)

Q. 8. Given the predicates is a (Person, Mammal)
(instance (Pee-Wee, Person)
(team (Pee-Wee, ABC)
uniform color (Pee-Wee, Green)
Express the above in the form of semantic or associative network.

Ans.

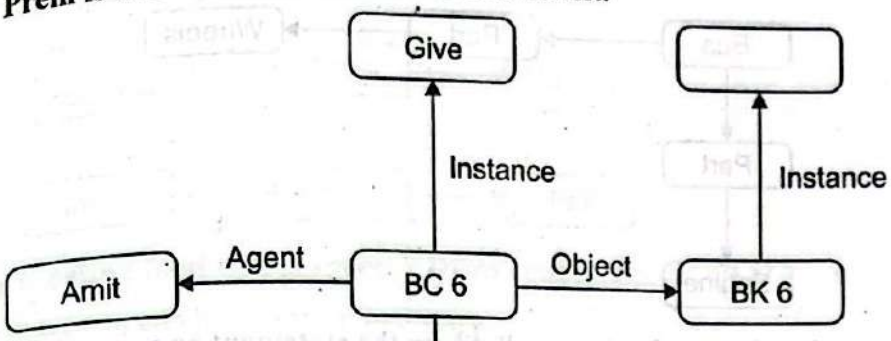


Q. 9. Represent the following as a semantic network.

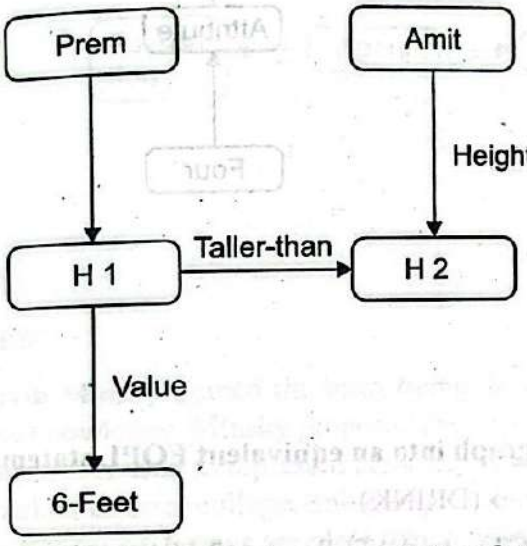
(a) Amit gave the book to Mersa.

(b) Prem is 6 feet tall and he is taller than Amit.

Ans. (a)



(b)



Q. 10. Construct a conceptual representation for the sentence, "Every car has an engine"

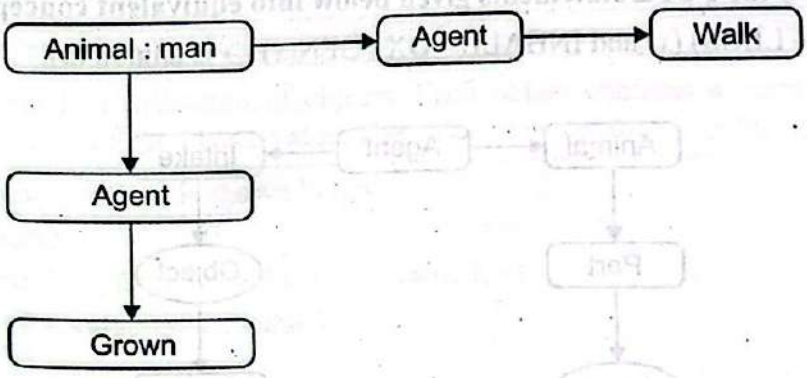
Ans. The conceptual graph representation is

[CAR : ∀] → (PART) → (ENGINE)

Q. 11. Transform the FOPL statement equivalent conceptual graph.

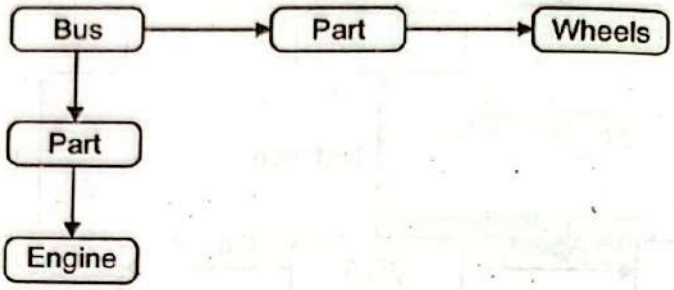
$\forall x$ Normal (x) and Grown (x) → Walk (x).

Ans. The conceptual graph is as follows.



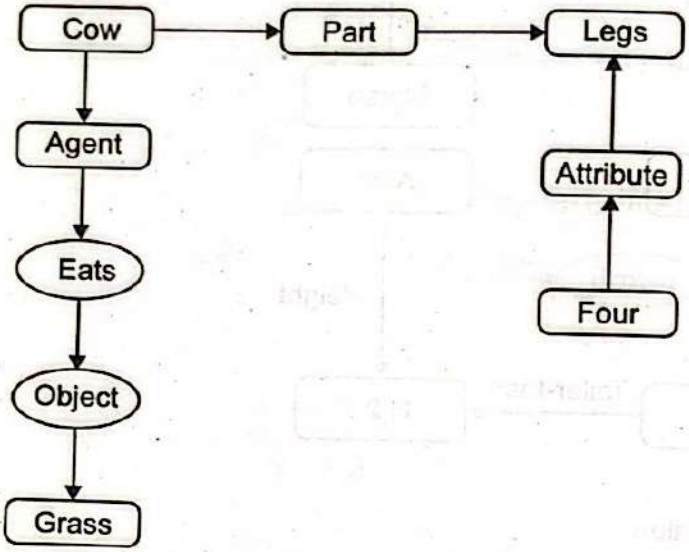
Q. 12. Give conceptual graph for the statement.—"Every bus has wheels and engine".

Ans.



Q. 13. "A cow has four legs and eats grass". Show the statement as a conceptual graph.

Ans.



Q. 14. Convert the following conceptual graph into an equivalent FOPL statement.

[CAMEL : clyde] → (Agent) → (DRINK)
→ [(OBJECT) → (WATER) → (ATTRIBUTE)
→ [50 – GALLONS].

Ans. E(x) for x is a camel.

D(x) for x is drinks water.

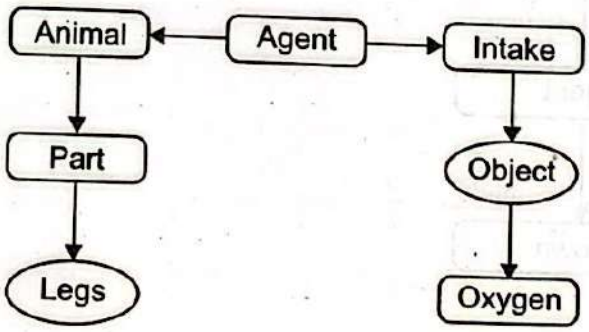
G(x) for drinks 50 gallons of water.

$\therefore$ $\exists x\,(E(x)$: clyde) & $D(x) \rightarrow G(x))$

Q. 15. Transform the FOPL statements given below into equivalent conceptual graph.

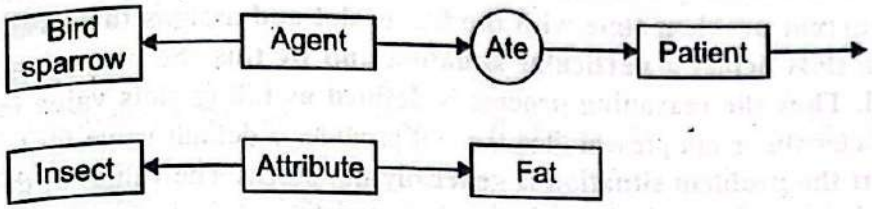
$\forall x$(HAS – LEGS) (x) and INHALE – OXYGEN(x) → is animal (x).

Ans.



Q. 16. Draw conceptual graph of the statement—"Sparrow ate a fat insect".

Ans. The conceptual graph for the statement is as follows.



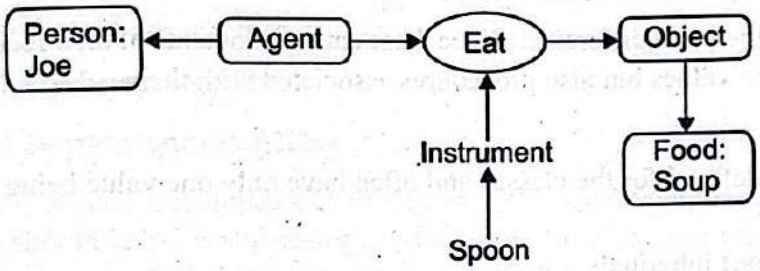
Q. 17. "Joe is eating food with a spoon." Draw conceptual graph of the statement.

Ans. The statement can be shown in two ways.

1. **Linear form of conceptual graph**

   [PERSON : joe] → (Agent) → [EAT] → (OBJECT) → [FOOD : soup] – (INSTRUMENT) → [SPOON]

2. **Conceptual graph representation**



4.5.2 Frames

In 1975, Marvin Minsky coined the term frame. It is a method used to represent conceptual and Commonsense knowledge. Minsky proposed the organisation of knowledge in form of small packets called as **frames**. It is used to represent activities associated with conceptual events taking place, in say, a supermarket, theatre, college, university etc. This knowledge in form of frame is stored in the KB. The contents of the frames are slots which have certain values. The slots may be of any type and of any size slots have names and values (or subfields) called as **facets**. Facets may also have names and any number of values. An appropriate frame is selected from the memory for reasoning. Frames are of 2 types —

(*a*) **Declarative frames:** Frames consisting of only descriptive type of knowledge.

(*b*) **Procedural frames:** A frame having knowledge about the action or procedures.

So, a frame is a collection of attributes or slots and associated values that describe some real world entity.

NOTE: Frames → have slots for various objects → slots have values

A **frame-system** is a collection of objects. Each object contains a number of slots. A slot represents an attribute. Each slot has a value. The value of an attribute can be another object.

A general frame structure is shown below:

(< frame name >
  (< slots 1 > (facet 1 < value 1 > ... value k >)
  (facet 2 < value 1 > ... value k >)
  ⋮
  (facet n <value n > ... <value k>)

The frames can be attached with another frame to create **network of frames. Reasoning, using frames is done by instantiation process. The process in which the given situation is matched with frames that are already existing is known as instantiation. The reasoning process tries to match the current problem state with the frame slot and assigns them values. The values assigned to the slots depict a particular situation and by this the reasoning process moves towards a goal. Thus the reasoning process is defined as filling slots value in frames.** If the given slot characteristic is not present then the slot provides a default value for the characteristic. **Please note that the problem situation is generally not static.** The values of the corresponding slots are updated if there is any deviation in the characteristic.

Types of Slots

Slots are of three types as follows:

I. Inherent slots

(*a*) These slots are specified by the use for each class and the values are given during the instantiating of this class to obtain an object.

(*b*) These are comparable with the recording field of data and hence are the characteristic of the knowledge under consideration.

(*c*) However, there is a difference. Since these can exist outside of their recording they can have one or more values but also procedures associated with them.

II. Meta slots

(*a*) These are defined for the classes and often have only one value being related to the given class.

(*b*) These are **not** inherited.

(*c*) Such an attribute is the **ako** (a – kind– of) link mentioned in the semantic networks which relates one class to a more general class.

(*d*) Each class is related by **ako** to at least one other class. There exists exactly one class linked to itself, which is the vertex of the graph of **ako** links.

(*e*) The slot **ako** determines one of the fundamental interests of the object representation — INHERITANCE.

(*f*) At the level of instanced frames, it allows (by default) a frame to inherit certain values of the classes to which it is bound (related) by **ako**. If this frame is bound to several others, there can be **multiple** inheritance. The value of **ako** is a set of frames.

III. Instantiated slots

(*a*) Are defined for objects which are instances of classes.

(*b*) As priori, their creation is automatic during instantiating since they are ascribed to the instantiated class.

(*c*) This type of attribute is the **isa link** mentioned for semantic networks.

Please note that each slot possesses an arbitrary number of facets. These are the declarations or procedures associated to the attributes.

Many languages like LISP (i.e. List Processing) has in-build procedures to create, access, modify, update and display frames.

We must keep the following points in mind—

1. Instances of SLOT are slots.
2. Associated with SLOT are attributes that each instance will inherit.
3. Each slot has a domain and a range.

4. Since slot is a set of all slots, so can be represented by metaclass called slot.
5. To compute attribute involves a procedure to compute its value.

Advantages of Frames

1. Are very flexible.
2. Are similar to human knowledge organization, easier to understand logic or rules.

Disadvantages of Frames

1. No standards.
2. Very generalized approach
3. No associated reasoning mechanism.

Problems with both semantic nets and frames

1. Both net and frames are weak slot and filler structures. Both are arbitrary.
2. Both are useful for representing certain sorts of knowledge.
3. Both lack semantics i.e., precise meaning.
4. Logic generalizes these schemes.
5. Syntax of KR scheme is irrelevant.

4.5.3 Conceptual Dependencies (CDs)

In 1977, Roger *C* Schank developed CD structures. CD is used to represent knowledge again. This representation should be powerful enough to represent these concept of sentences of natural language. **CD theory states that different sentences which have same meaning should have some unique CD representation.** In CD theory, 5 different types of ontological (state of being) building blocks are distinguished —

(*a*) Entities.

(*b*) Actions.

(*c*) Conceptual cases.

(*d*) Conceptual Dependencies.

(*e*) Conceptual tenses.

Entities

(*a*) Picture Producer (PP) are actors that perform different acts.

(*b*) Picture Aiders (PA) are supporting properties or attributes of producers.

(*c*) ACT: Actions done by an actor to an object.

(*d*) LOCs: Shows the location of action,

Actions

(*a*) Primitive actions (ACTs).

(*b*) **Action Aiders** *(AA)* are properties or attributes of primitive action.

For **primitive actions,** a typical CD set is —

ATRANS	Transfer (example., 'give')
INGEST	Ingest into the body (example. 'eat')
EXPEL	Expel from body (example. 'cry')
ATTRANS	Abstract transfer (example. 'owner ship')

MORE	Move a body part (example. 'kick')
GRASP	Grasp an object (example. 'clutch')
PROPEL	Throw an object (example. 'push')
PTRANS	Physical transfer (example. 'go')
MTRANS	Mental transfer (example. 'tell')
CONC	Conceptualize (i.e., think about something).
ATTEND	Direct sensory orange to observe (example. listen).

The main goals of this theory are —

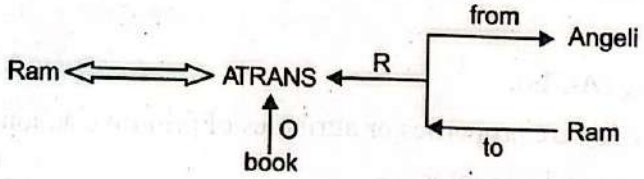
1. To capture the implicit concept of a sentence and make it *explicit*.
2. To help in drawing inferences from sentences.
3. For any 2 or more sentences that are identical in meaning, there should be only one representation of meaning.
4. To develop language conversion packages.
5. To provide a means of representation which are language independent.

In addition to the actors and objects other concept of time, location, source and destination are also used in *CD* representation—

(*i*)	O	:	Object case relationship
(*ii*)	r/R	:	Recipient case relationship
(*iii*)	p	:	Part
(*iv*)	f	:	future
(*v*)	t	:	transition
(*vi*)	Is	:	Start transition
(*vii*)	Tf	:	finish transition
(*viii*)	k	:	continuing
(*ix*)	?	:	interrogative
(*x*)	/	:	negative
(*xi*)	nil	:	present
(*xii*)	delta	:	timeless:
(xiii)	C	:	Conditional.

For example 1: Ram gives Anjali a book.

Its *CD* structure is as follows—



where the symbols used are —

(*a*) **Arrow:** indicates the direction of dependency.

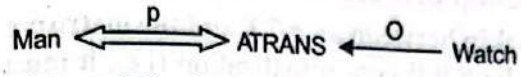
(*b*) **Double arrow** indicates two-way link between actor and actions.

(*c*) **ATRANS** is a primitive to describe this event.

(d) *O* represents object case relationship.

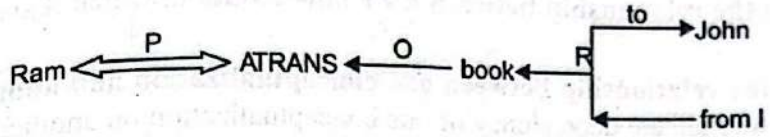
(e) *R* indicates recipient case relationship.

***Example 2:* The man took a watch**



Here, 'men' is an actor and 'watch' is the object of the action 'took'.

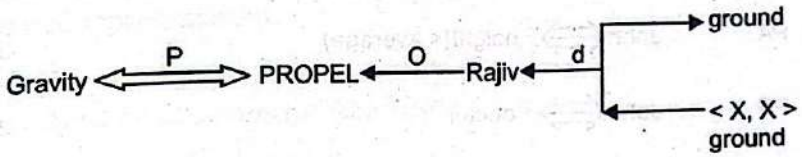
***Example 3:* I gave John a book**



Many a times, actors are missing from a sentence. This is a challenge. *For example.* 'Rajiv fell', sentence has a missing actor as another force "gravity" was acting. In such cases, we need additional optional notations :

R = reason
r = results
i = intends
d = direction
I = Instrument
R = recipient, state change.

So, the sentence '*Rajiv fell*' in CD is as follows—



Parsing a sentence into CD representation is analogous to the process of parsing using a case grammar. The basic algorithm for building CD structure is as follows—

S1 : Obtain the root lexical item.

S2 : Access the lexical entry for the item and obtain the association tests and actions.

S3 : Perform the specified actions given with the entry.

We must study some rules for CD representation.

Rules of **CD**

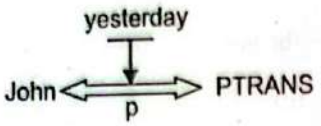
Rule 1 **describes the relationship between an actor and the event he or she causes.** This is a two-way dependency since neither actor nor event can be considered primary. The letter above the dependency link indicates past tense.

Rule 2 **describes the relationship between PP and PA** that is being asserted to describe it. Many state descriptions, such as height, are represented in CD as numeric scales.

Rule 3 **describes the relationship between two PPs**, one of which belongs to the set defined by the other.

Rule 4 **describes the relationship between a PP and an attribute** that has already been predicated of it. The direction of the arrow is towards the PP being described.

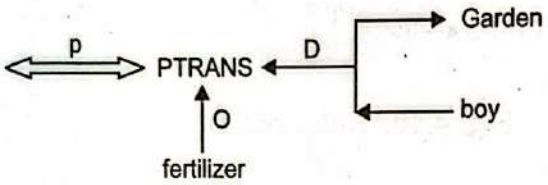
Rule 5 **describes the relationship between two PPs one of which provides a particular kind of information about the other.** The three most common types of information to be provided in this way are procession (shown as POSS-BY), location (shown as LOC), and physical containment (shown as CONT). The direction of the arrow is again toward the concept being described.

(a)


(b)


(c)


(d)


(e)


Q. 2. Give the CD representation of the following sentences.

(*i*) Joe gave Sue a flower.

(*ii*) Mohan will eat pizza with fork.

(*iii*) Charlie drove the car.

(*iv*) Since smoking can kill you, I stopped.

(*v*) Bill is a programmer.

(*vi*) Mary goes.

(*vii*) Mary will go.

(*viii*) Neeraj cried.

(*ix*) Bill threatened John with a broken nose.

(*x*) Anil punched Ram.

(*xi*) Ram lied.

(*xii*) Joe told Lucy he would not go to movies with her. Her feelings were hurt.

(*xiii*) Prem went to the park with peacocks.

(*xiv*) John will eat noodles with fork and knife.

(*xv*) Mohan is a sales manager.

(*xvi*) Kapil ate some soup.

(*xvii*) Mohan gave some books to Reeta.

(*xviii*) He will speak.

(*xix*) Anil will go tomorrow.

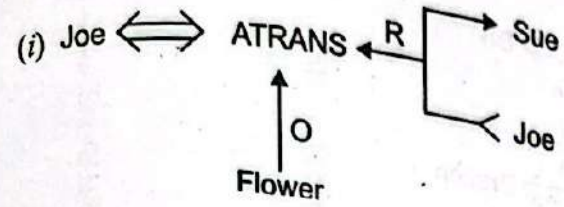
(*xx*) While going home, he saw a peacock.

(*xxi*) Ram heard noises in the kitchen.

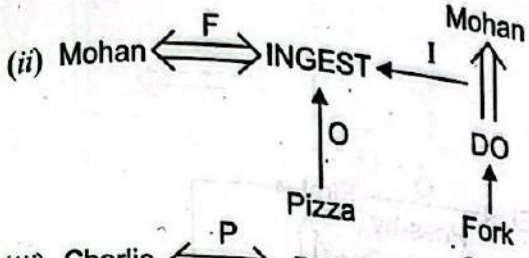
(*xxii*) Hair is short.

(*xxiii*) Hari is a nice boy.

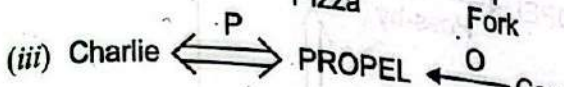
Ans. (*i*)



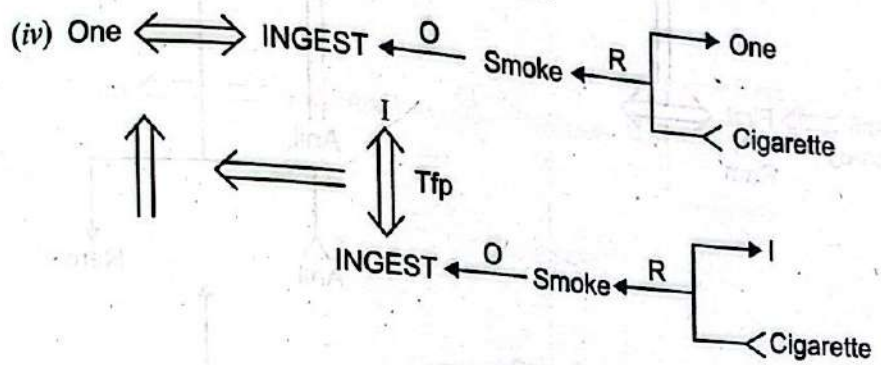
(*ii*)



(*iii*)



(*iv*)



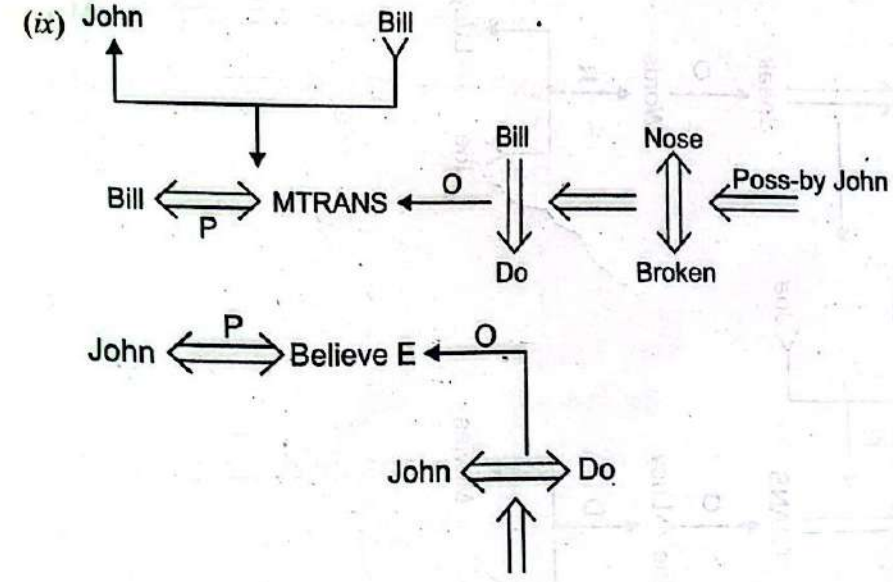
(*v*) Bill ⟺ Programmer

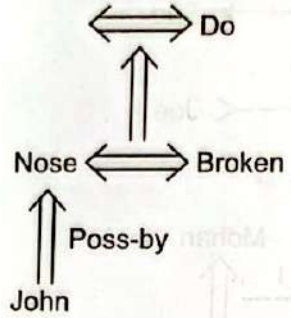
(*vi*) Mary ⟺ PTRANS

(*vii*) Mary $\overset{F}{\Longleftrightarrow}$ PTRANS

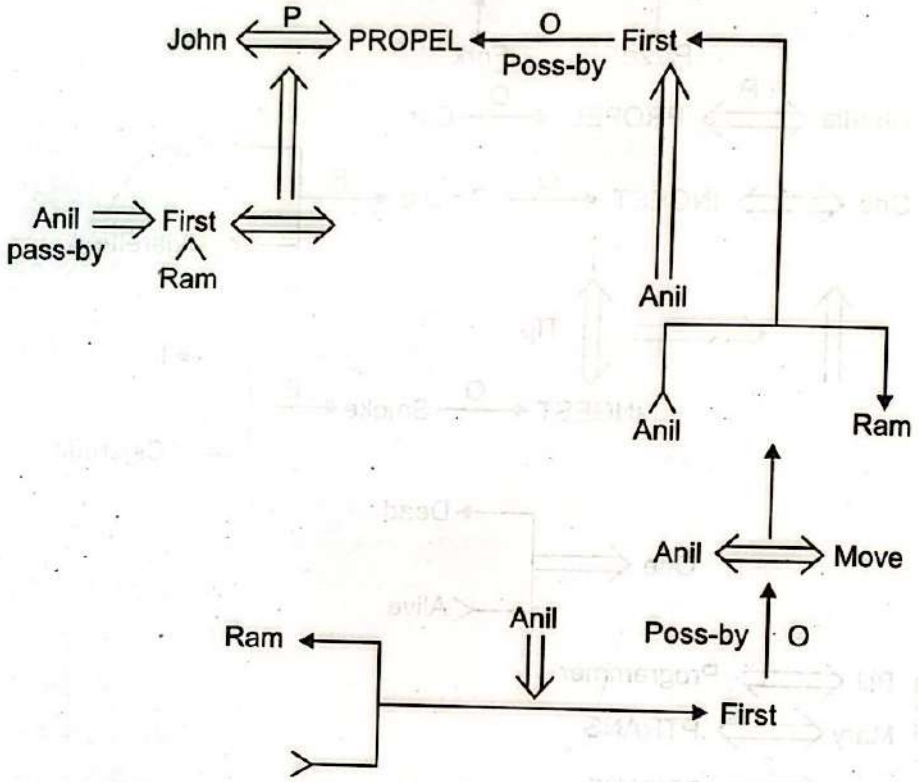
(*viii*) Neeraj $\overset{P}{\Longleftrightarrow}$ EXPEL

(*ix*)

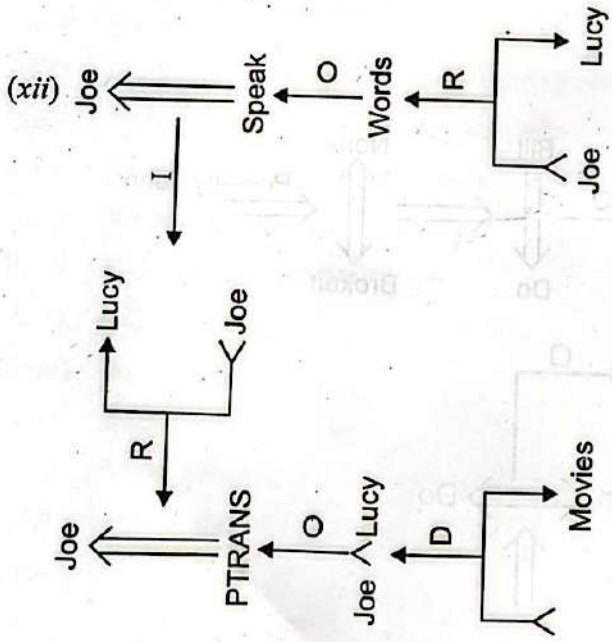


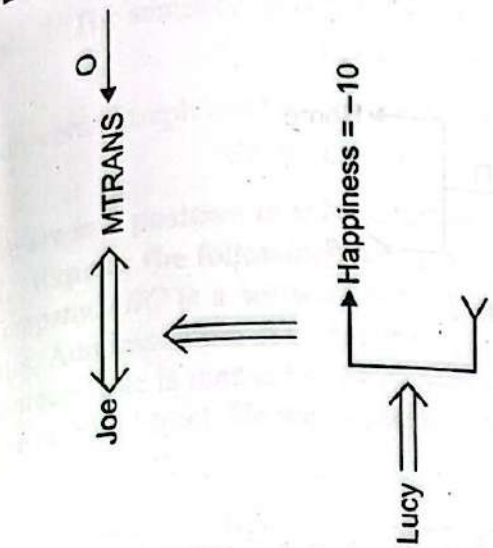


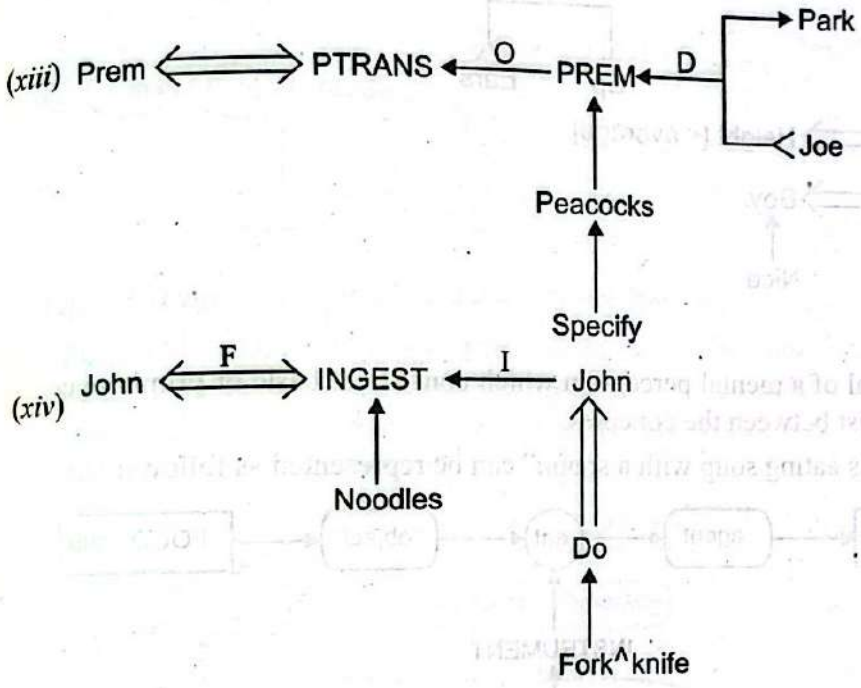
(x)

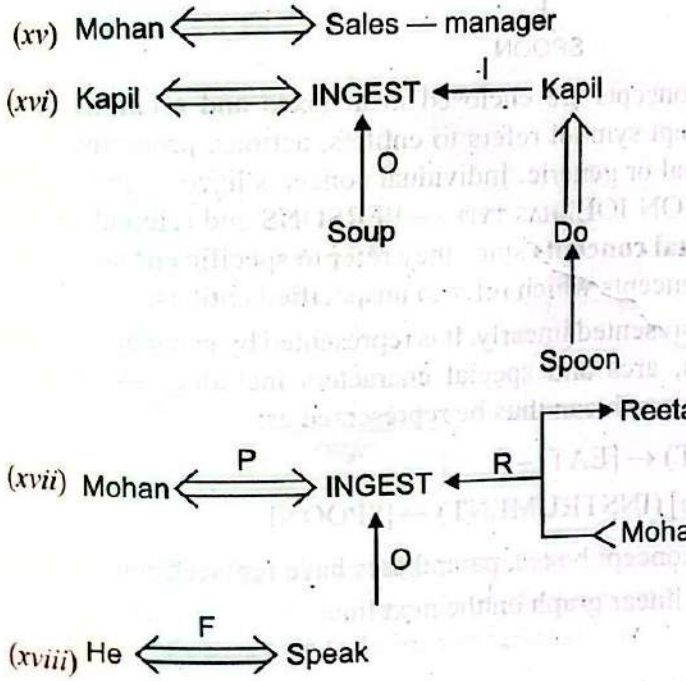


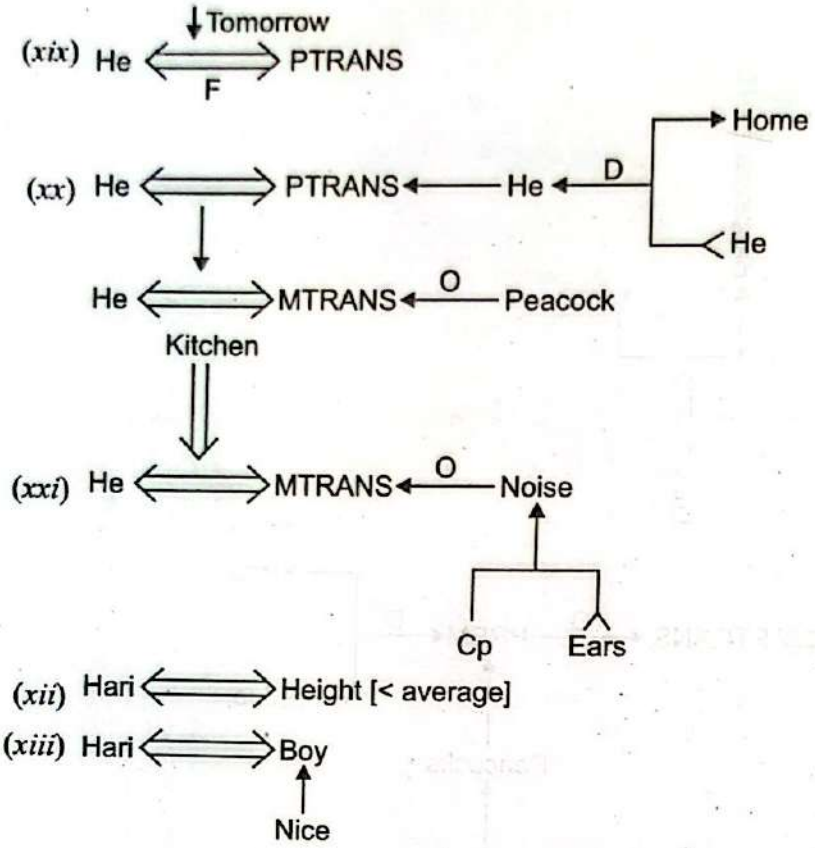
(xi) Ram P MTRANS







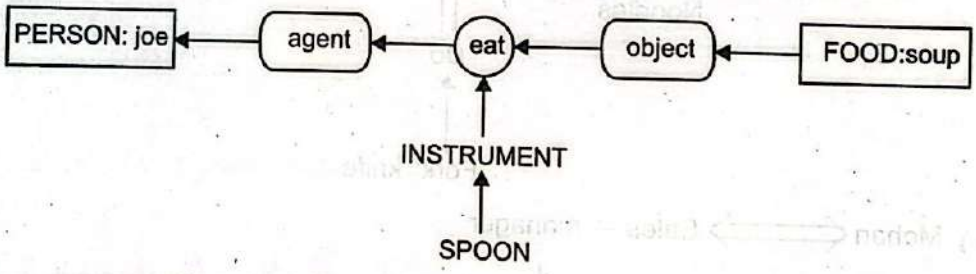




Conceptual graphs

It is a graphical portrayal of a mental perception which consists of basic or primitive concepts and the relationships that exist between the concepts.

For example: "Joe is eating soup with a spoon" can be represented as follows:



This is a conceptual graph. Concepts are enclosed in a boxes and relations between the concepts are enclosed in ovals. Concept symbol refers to entities, actions, properties or events in the world. A concept may be individual or generic. Individual concepts have a type field followed by a referent field. The concept PERSON JOE] HAS TYPE — PERSONS and referent Joe. Referents like Joe and a soup are called **individual concepts** since they refer to specific entities. EAT have no referent field since they are generic concepts which refer to unspecified entities.

A conceptual graph can also be represented linearly. It is represented by using upper & lowercase characters of alphabet, boxes circles, arcs and special characters including —, ?, !, *, #, @, ∀, ~, [,], (,), →, ←, } & }. The above graph can thus be represented as:

[PERSON : Joe] ← (AGENT) ← [EAT] –

(OBJECT) → [FOOD : Soup] (INSTRUMENT) → [SPOON]

where square brackets have replaced concept boxes, parentheses have replaced relation circles. The dash signifies continuation of the linear graph on the next line.

Example. 2. The sentence "Every car has an engine" can be represented as a conceptual graph —

$$[CAR : \forall] \rightarrow (PART) \rightarrow [ENGINE]$$

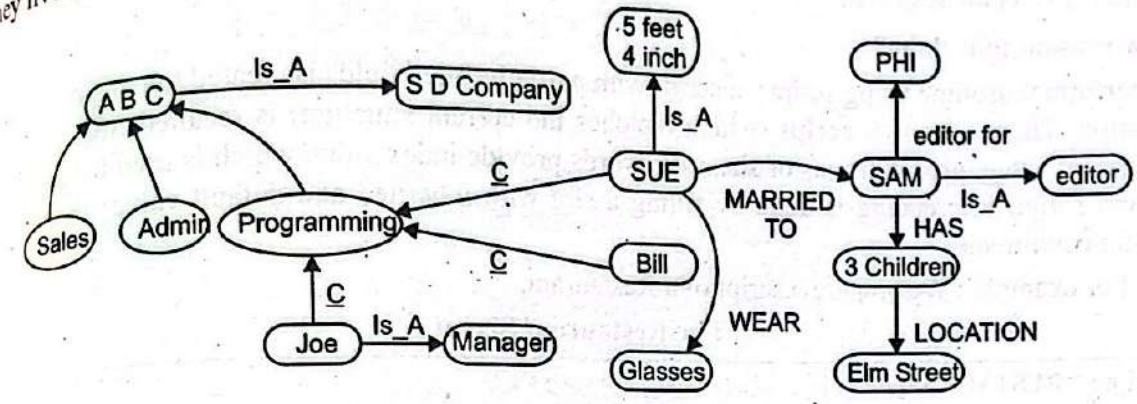
This conceptual graph can be converted into its FOPL as follows:

$$\forall x \, \exists y \, (CAR \, (x) \rightarrow (ENGINE \, (y) \, \& \, PART \, (x, y)))$$

We are in a position to solve examples now.

Q. 1. Express the following concepts as an associative network structure —

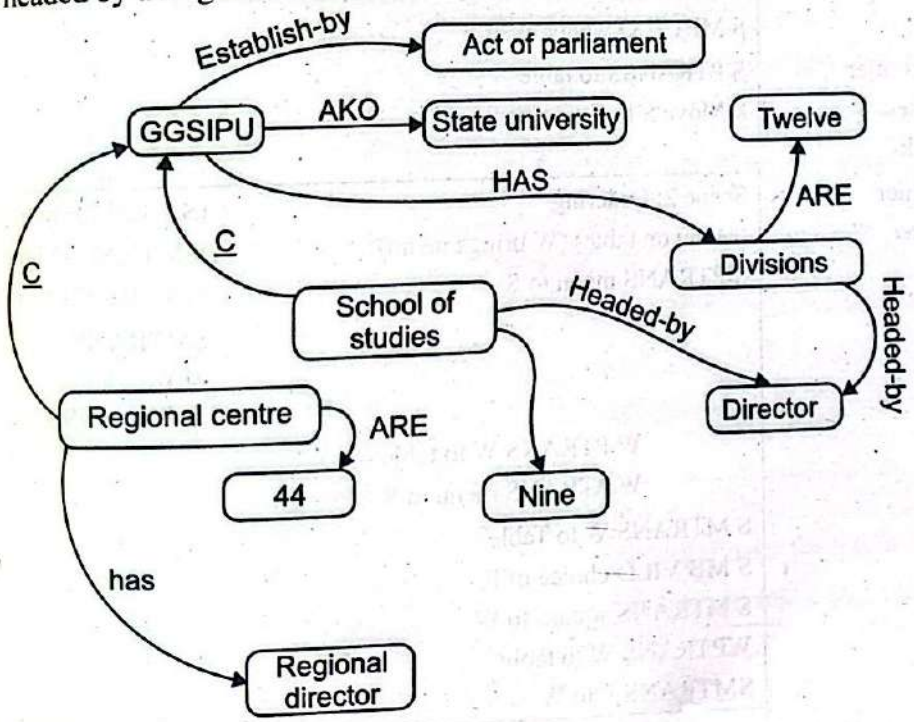
Company *ABC* is a software development company. Three departments within the company are Sales, Administration & Programming. Joe is the manager of Programming. Bill and Sue are programmers. Sue is married to Sam. Sam is an editor for Prentice Hall. They have 3 children and they live on Elm Street. He wears glasses and is 5 feet 4 inches tall.



Q. 2. Represent the following as an associative network.

"GGSIPU is a State University established by the Act of Parliament. It has 9 schools of studies and 12 divisions to support the academic activities at the headquarters. It has 44 regional centres spread over the whole country. Each school and division is headed by a Director and each regional centre is headed by a Regional Director".

Ans.



4.8.4 Scripts

A script is a structure that prescribes a set of circumstances which could be expected to follow on from one another. It is similar to a chain of situations which could be anticipated. Like a script for a play, the script's structure is defined in terms of —

(*a*) Actors (*b*) Roles (*c*) Props and (*d*) Scenes.

Please note that the slots in a script correspond to the parts of the event which are filled with CD primitives. It could be considered to consist of a number of slots or frames but with more specified robes. Scripts are frame-like structures to represent knowledge grouped from our common experiences like going to a movie, eating in a bar, visiting a doctor or shopping in a MALL etc.

Scripts are useful because —

(*a*) Events tend to occur in known runs or patterns.

(*b*) Entry conditions exist which allow an event to take place.

(*c*) Casual relationships between events exist.

(*d*) Prerequisites exist.

How reasoning is done?

To perform reasoning using scripts, a script with partially filled fields is created to meet the current situation. Then, a **known script** (which matches the current situation) is recalled from memory. The script name, preconditions or shear keywords provide index value which is used to search the current script. Inferencing is done by filling a slot with inherited and default values that satisfy certain conditions.

For example : We prepare a script of a Restaurant.

The Restaurant Script

Script : RESTAURANT Track : Coffee shop Propos : Tables Menu F = Food Check Money Roles: S = Customer W = Waiter C = Cook M = Cashier O = Owner	Scene 1: Entering S PTRANSS into restaurant S ATTEND eyes to tables S MBUILD where to sit S PTRANSS to table S Move S to sitting position	
	Scene 2: Ordering (Menu on table) (W brings menu) S PTRANS menu to S W PTRANS W to table W ATRANS menu to S S MTRANS W to Table S MB VILD choice of F S MTRANS squard to W WPTRANS W to table SMTRANS F to W	(S asked for menu) S MTRANS signal to W W PTRANS W to table SMTRANS 'need menu' to W W PTRANS W to menu
	W PTRANS W to C W MTRANS (ATRANS F) to C	

Entry conditions: S is hungry. S has money.	CMTRANS no F to W WPTRRANS W to S W MTRANS no F to S (go back to*) or (go to Scene 4 at no pay path) CDO (prepare F script) to Scence 3
Results: S has less money O has more money. S is not hungry. S is pleased (optional).	Scene 3 : Eating CATRANS F to ww ATRANS F to S S INGEST F Option: REturm to Scene 2 to order more; otherwise, go to scene 4
	Scene 4: Exiting S MTRANS to W (W ATRANS check to S W MOVE (write check) W PTRANS W to S W ATRANS check to S S ATRANS tip to W S PTRANS S to M S PTRANS S to out of restaurant (No pay path)

Let us write an other script now.

Script Name: Interaction with car salesperson

PROPS	Scenes
Car showroom Car Feature Booklet Money Time	**Scene 1** • Customer enters the showroom • Search for a particular brand. • Salesperson attends customer and welcomes him
ROLES Customer Salesperson Owner Cashier	**Scene 2** • Sales person shows various cars • Customer selects particular car • Salesperson describes its features
INITIAL • Customer needs a car • Customer has more money • Salesperson has details about car • Availability of car in showroom • Owner has less money	**Scene 3** • Customer asks for a test drive • Salesperson takes him on a drive
	Scene 4 • Customs is ready to buy • Customser asks for bill and warranty • Salesperson gets him warranty book and bill

RESULTS	Scene 5
• Customer bought a car • Salesperson got commission • Customer has less money • Owner has more money • Particular car count decreases	• Customer pays the money • Salesperson deposits it with cashier • Customer to gives commission to salesperson • Drives car back home

TRACKS none

Please note the following points about scripts:

1. If a particular script is to be applied then it must be activated and this activation depend on its significance.
2. If a topic is mentioned in passing then a pointer to that script could be held. Else if the topi is important then the script should be opened.
3. Having too many scripts is dangerous.
4. If events follow a known trail, then we can use scripts to represent the action involved an use them to answer detailed questions.
5. Different trails may be allowed for different outcomes of scripts.

Advantages of scripts

(*a*) They have ability to predict events.

(*b*) A single coherent interpretation may be built up from a collection of observations.

Disadvantages of scripts

(*a*) They are less general than framer.

(*b*) They are not suitable to represent all kinds of knowledge.

Applications of scripts

Shank and his team at YALE University used scripts in one system named as SAM — Script applier mechanism. SAM reads and reasons with text to demostrate an understanding of stories that were script based.

SUMMARY

This chapter discusses about advanced knowledge representation techniques like frames, nets, CG, CD and scripts. KR is one of the critical aspects to make our computer behave intelligently.

MULTIPLE CHOICE QUESTIONS [MCQs]

1. Data when processed gives information and Information when processed gives —

(*a*) Knowledge (*b*) Domain (*c*) Wisdom (*d*) None of these.

2. The knowledge about what we know is —

(*a*) Meta knowledge (*b*) Mini knowledge

(*c*) Micro knowledge (*d*) None of the above.

3. Concepts, objects, nets, logic etc are the examples of

(*a*) Tacit knowledge (*b*) Explicit knowledge

(*c*) Knowledge base (*d*) None of the above.

4. According to Robert Kowalski
   (a) Logic = Algorithm + Control (b) Algorithm = Logic + Control
   (c) Algorithm = Logic – Control (d) None of the above
5. The process of producing proofs by refutation or contradiction is known as —
   (a) Resolution (b) Proposition (c) Unification (d) None of these.
6. The universal quantifier is —
   (a) ∃ (b) ∧ (c) ∀ (d) =
7. The process of eliminating existential quantifiers is known as
   (a) Resolution (b) Skolemisatio (c) Unification (d) None of these.
8. The Recursive process of merging two or more predicates with constants or with some variables is known as —
   (a) Resolution (b) Proposition (c) Unification (d) Proposition
9. Forward chaining is a —
   (a) Data-driven search (b) Goal-driven search
   (c) Heuristic search (d) None of the above.
10. A CD primitive that is used to do mental transfer is –
    (a) ATRANS (b) ATTRANS (c) PTRANS (d) MTRANS

ANSWERS

1. (a)	2. (a)	3. (b)	4. (b)	5. (a)
6. (c)	7. (c)	8. (c)	9. (a)	10. (d)

CONCEPTUAL SHORT QUESTIONS WITH ANSWERS

Q. 1. What is chaining?

Ans. In inferencing process, the rules are applied in succession till a situation arrives that there is no rule which is applicable. It is called as **chaining.**

Q. 2. What do you mean by a Universe of Discourse?

Ans. The universe of discourse or domain of discourse indicates the following —

(a) A set of entities that quantifiers deal with.

(b) Entities can be a set of integers, a set of real numbers, a set of students etc.

Universe is the domain of individual variables. Even propositions in logic are statement of an object of a universe.

Q. 3. What is matching? Name some matching techniques.

Ans. In matching process, the LHS of rule is compared with the input and all rules that match are extracted. Actually, both the current state of the problem and its preconditions are matched with the LHS of the rule. This is known as **matching**. Some matching techniques are as follows —

(a) Indexing

(b) Matching with variable

(c) Complex matching variable.

(c) All courses in the basket weaving department are easy

(d) BK301 is a basket weaving course.

Use resolution to answer the questions— "What course would Steve like"?

[MDU, BE (CSE)-8th Sem., Dec. 20

Ans. Convert the given facts into predicates firstly –

1. like (steve, easy course)
2. hard (Science, course)
3. $\forall x \forall y$ [courses $(x, y) \rightarrow$ easy (x, y)]
4. Courses $(x, y = BK301)$

Now, we convert these predicates into clausal form—

1. $\forall x \forall y$ $[\text{course}(x, y) \rightarrow \text{easy}(x, y)]$

$\forall x \forall y$ $[\neg \text{course}(x, y) \vee \text{easy}(x, y)]$

$$\begin{bmatrix} \because \neg(\neg P) = P \\ \& \neg(a \wedge b) = \neg a \vee \neg b \\ \& \neg(a \vee b) = \neg a \wedge \neg b \end{bmatrix}$$

2. Course $(x, y = \text{BK } 301)$

$[\forall x \forall y (\neg \text{course}(x, y) \vee \text{easy}(x, y)] \wedge \text{course}(x, y = \text{BK } 301)$

$\Rightarrow$ easy $(x, \text{BK } 301)$ ...(3

This implies that BK301 is an easy course.

Now, consider equation (1) and equation (3):

Like (steve, easy course) $\wedge$ easy (x, BK301)

$\therefore$ Steve likes BK301

Now, consider (1) and (2)

Like (steve, easy course) $\wedge$ hard (Science, courses)

$\Rightarrow$ steve doesn't like science

What course would Steve like?

It is clearly proved by equation (3) that Steve likes BK301.

Q.9 Given two clauses— Loves (Father (a), a) and loves (y, x). What must be generated as a result of resolving these two clauses? **[MDU, BE (CSE)-8th Sem., Dec 2005]**

Ans. Given:

loves (Father (a), a) and loves (y, x)

a = x

Father (a) = y

Father (x) = y

This means that Father of x is y.

Now, love (Father (a), a)

loves (y, x) ∨ loves (x, y)

loves (Father (a), a)

loves (y, x)

Father (a) = y

∴ and a = x

loves (Father (a), a)

loves (x,y)

∴ Father (a) = x

and a = y

Which is NOT possible.

Thus, these two clauses cannot be unified.

Q. 10. The following statements are given —

(*i*) Rishi likes to eat peanut.

(*ii*) Rishi eats whatever John eats.

(*iii*) John eats only eatables.

(*iv*) Anybody who eats eatables is a human being.

(*v*) Peanut is an eatable.

(*a*) Convert these into wff and clausal form.

(*b*) Is John a human being? Answer this using back tracking and resolution.

[MDU. BE (CSE)-8th Sem., May 2005]

Ans. (*a*) The WFF notations are—

1. Likes (Rishi, Peanut)
2. ∀xy eats (Rishi, y) → eats (John, y)
3. ∀y eats (John, y)
4. ∀x ∀y eats (x, y) → human being (x)
5. eatable (y = peanut)

Its clausal form is as follows —

1. Likes (Rishi, y1) ...(1)
2. ∀y eats (Rishi, y) → eats (John, y)

   ∀x ¬ eats (Rishi, y) ∨ eats (John, y)

   ∴ ¬ eats (Rishi, y2) ∨ eats (John, y2) ...(2)
3. ∀y eats (John, y) is

   eats (John, y3) ...(3)
4. ∀x ∀y eats (x, y) → human being (x)

   ∀x ∀y ¬ eats (x, y) ∨ human being (x)

   ∴ ¬ eats (x1, y4) ∨ human being (y4) ...(4)
5. eatable (y = peanut) is

   eatable (y5) ...(5)
6. Is John a human being?

   i.e. human being (John)

We start with opposite of this (i.e. negation)

¬ human being (John)

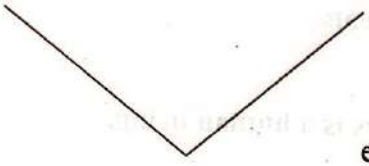
∴ Using back tracking and Resolutions we get —

human being (John) (4)

x1 = John

x4 = eatable

eats (John, eatable) (3)



eats (John eatables)

Nil

∴ It is proved that John is a human being.

Q. 11. Assume the following facts —

(*a*) Ram likes only easy courses.

(*b*) Engg. courses are hard.

(*c*) All courses in Arts are easy.

(*d*) AR04 is an Arts course.

Use resolution to answer the question, "What course would Ram like?"

[MDU, BE (CSE)-8th Sem., Dec. 2004]

Ans. Let us find its Predicate clause firstly,

$\exists y$ easy (y) $\rightarrow$ likes (Ram, y)

$\exists y$ easy course (x) $\rightarrow$ hard (x)

$\forall x$ art course (x) $\rightarrow$ easy (x)

art course(AR04)

∴ Its clausal form is—

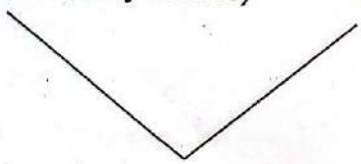
like (Ram, easy courses)

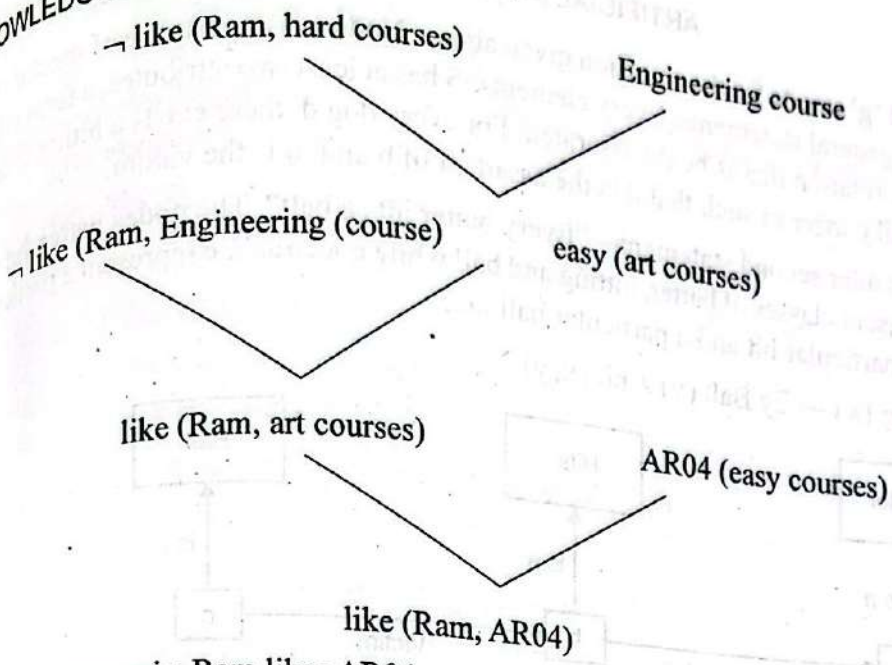
hard course (Engineering)

easy (art courses)

AR04 (art course). Now, we use resolution to prove our query.

like (Ram, easy courses) ¬ hard courses (Engineering)





∴ Answer is: Ram likes AR04.

Q. 12. **Describe the term 'Partitioned Semantic Net'. Also construct a Partitioned semantic Net representation for the following:**

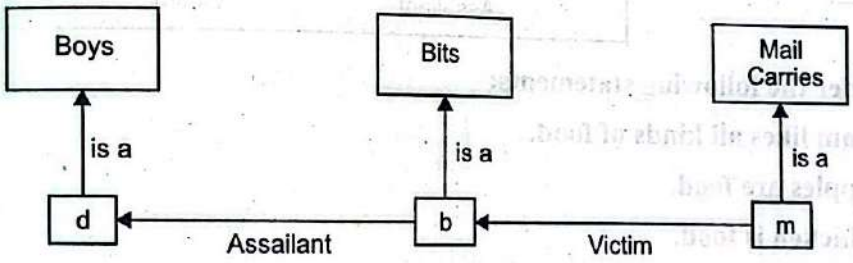
(*a*) Every dog has bitten a mail carrier.

(*b*) Every batter hits a ball.

[MDU, BE (CSE)-8th Sem., Dec. 2004]

Ans. It is a representation for simple quantified expressions in a semantic net in which semantic net is partitioned into a hierarchical set of spaces each of which corresponds to the scope of one or more variable.

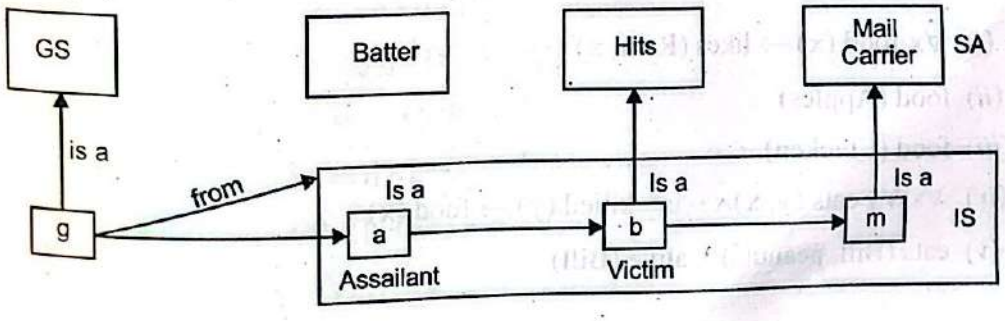
(*a*) Consider a simple net corresponding to statement—"The dog bit the mail carrier", given below —



The nodes–Dogs, Bite and bitings and Mail carriers. Nodes d, b and m represent a particular dog, a particular biting and a particular Mail carrier. This fact can easily be represented by a single net with no partitioning. If we represent the fact "Every dog has bitten a mail carrier" or the logic.

$\forall x \text{ Dog}(x) \rightarrow \exists y \text{ Mail} - \text{carrier } (y) \wedge \text{Bite } (x, y)$

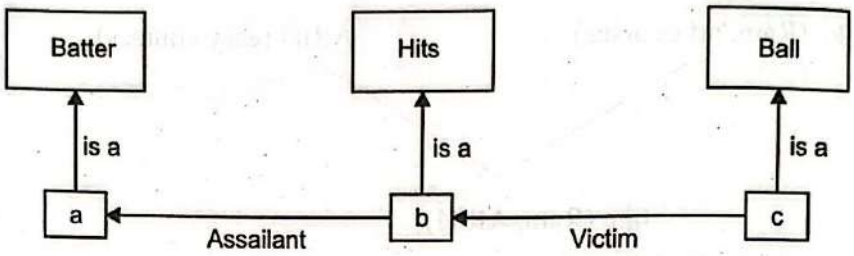
Example,



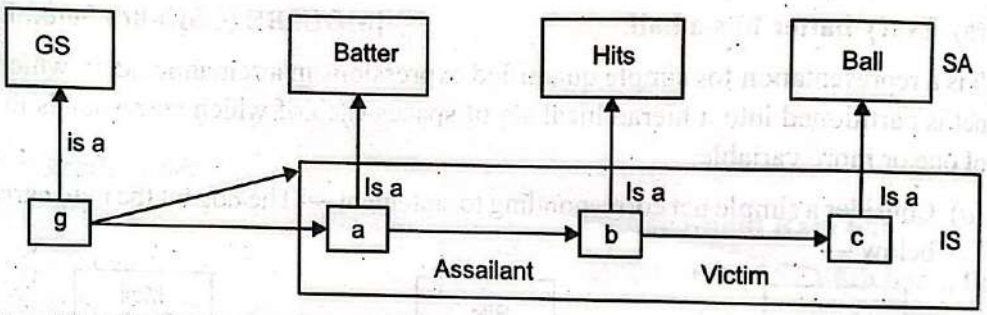
The node 'g' stands for the assertion given above. Node g is an instance of special cla of GS of general statements. Every element GS has at least two attributes, a form whi states the relation that is being asserated. For every dog d, there exists a biting event and a mail carrier m such that d is the assaitant of b and m is the victim.

(*b*) Now, consider second statement—"Every batter hits a ball". The nodes batter, hit an ball represent classes of batter, hitting and ball while nodes a, b, c represent a particul batter, a particular hit and a particular ball i.e.,

$\forall x$ Batter $(x) \rightarrow \exists y$ Ball $(y) \wedge$ hit (x, y)



To represent this face, we take node g which stands for assertion given above. Node is an instance of the special class GS of general statements. Every element of GS ha at least two attributes, a form which states the relation that is being asserted and one o more $\forall$ connections.



Q. 13. Consider the following statements:

(*i*) Ram likes all kinds of food.

(*ii*) Apples are food.

(*iii*) Chicken is food.

(*iv*) Anything anyone eats and isn't killed by is food.

(*v*) Bill eats peanuts and is still alive.

(*vi*) Rita eats everything Bill eats.

Translate these sentences into formulas in predicate logic.

[MDU, BE (CSE)-8th Sem., & RTM, Nagpur Univ. BE (IT), Winter 2007 & 2008]

Ans.

(*i*) $\forall x$ food $(x) \rightarrow$ likes (Ram, x)

(*ii*) food (Apples)

(*iii*) food (Chicken)

(*iv*) $\forall x \forall y$ eats $(y, x) \wedge \sim$ is_killed $(y) \rightarrow$ food (x)

(*v*) eats (Bill, peanuts) $\wedge$ alive (Bill)

(vi) $\forall x$ eats (Bill, x) $\rightarrow$ eats (Rita, x)

Q. 14. Say, whether or not the following pairs of expressions are unifiable and show the most general unifier for each unifiable pair —

(a) P(x, B, B) and P(A, y, z)

(b) **P (g (f(v)), g(u)) and P(x, x).**

(c) **P (x, f(x0) and P (y, y)**

(d) **P (y, y, B) and P (z, x, z)**

(e) **2 + 3 = x and x = 3 + 3**

Ans. (a) Yes, is unifiable. The most general unifier is —

A / x, B / y, B / z

(b) Yes". Unifier is g(u)/x, f(v)/u

(c) No. It doesn't unify.

(d) Yes. B/z, B/y, B/x

(e) Yes. First replace x by y in 2nd expression (standardizing variables) then (2 +3)/y and (3 + 3)/x.

Q. 15. Explain why P (f (x, x), A) does not unify with P (f (y, f (y, A)), A).

Ans. The problem involves an attempt to unify f (x, x) with f (y, f (y, A)). In these expressions, the outer f's match and y can be substituted for x leaving the problem of unifying y with f (y, A). Here, since y occurs in f (y, A), these two expressions cannot be unified.

Q. 16. Convert the following to clause form:

1. $((\exists x)[P(x)] \vee ((\exists x)[Q(x)]) \supset (\exists x)[P(x) \vee Q(x)]$
2. $(\forall x)[P(x) \supset (\forall y)[(\forall z)[Q(x,y)] \supset \neg(\forall z)[R(y,x)]]]$
3. $(\forall x)[P(x) \supset (\exists x)[(\forall z)[Q(x,y)] \vee (\forall z)[R(x,y,z)]]$
4. $(\forall x)[P(x) \supset Q(x,y)] \supset ((x\ y)[P(y)] \wedge (\exists z)[Q(y,z)])$

Ans.

1. $((\exists x)[P(x)] \vee ((\exists x)[Q(x)]) \supset (\exists x)[P(x) \vee Q(x)]$

$(\neg((\exists x)[P(x)] \vee (\exists x)(Q(x)) \vee (\exists x)[P(x) \vee Q(x)]$

$(\neg((\exists x)[P(x)] \wedge \neg(\exists x)(Q(x)) \vee (\exists x)[P(x) \vee Q(x)]$

$((\forall x)[\neg P(x)] \wedge (\forall x)!\neg Q(x) \vee (\exists x)[P(x) \vee Q(x)]$

$((\forall x)[\neg P(x)] \wedge (\forall x)!\neg Q(y) \vee (\exists x)[P(z) \vee Q(z)]$

$((\forall x)[\neg P(x)] \wedge (\forall y)[\neg Q(y)] \vee [P(Sk) \vee Q(Sk)]$

$(\forall x)(\forall z)[[\neg P(x)] \wedge Q(y)] \vee [P(Sk) \vee Q(Sk)]]$

$[\neg P(x) \wedge Q(y)] \vee [P(Sk) \vee Q(sk)]$

$[\neg P(x) \wedge P(Sk) \vee Q(Sk)] \wedge [\neg Q(y) \vee P(Sk) \vee Q(Sk)]]$

$[\neg P(x) \wedge P(Sk), Q(Sk)][\neg Q(z) \vee P(Sk) \vee Q(Sk)]$

2. $(\forall x)[P(x) \supset (\forall y)[(\forall z)[Q(x,y)] \supset \neg(\forall z)[R(y,x)]]]$

3. $P(x) \vee \neg P(y) \vee B(y)$
4. $P(01)$
5. $\neg P(02)$
6. (negation of theorem): $\neg G(x)$
7. (resolving 6 and 2b yields) $B(x)$
8. (resolving 7 and 1b yields) $P(x) \vee G(x)$
9. (resolving 8 and 5 yields) $G(02)$
10. (resolving 9 and 6 yields) contradiction.

Q. 19. The function cons(x,y) denotes the list formed by inserting the element x at the made of the list y. We denote the empty list by Nil; the list (2) by cons (2, Nil); the list (1, 2 by cons (1, cons (2, Nil)); and so on. The formula Last (1, e) is intended to mean tha i.e. the last element of the list 1. We have the following axioms:

- $(\forall u)\,[\text{Last}(\text{cons}(u, \text{Nil}), u)]$
- $(\forall x\ y\ z)\,[\text{Last}(y, z) \supset \text{Last}(\text{cons}(x, y), z)]$

1. Prove the following theorem from these axioms by the method of resoltion refutation $(\exists v)[\text{Last}(\text{cons}(2, \text{cons}(1, \text{Nil})), v)]$
2. Use answer extraction of find v, the last element of the list (2, 1).

Ans. 1. We first write the axioms in clause form:

(*a*) $\text{Last}(\text{cons}(u, \text{Nil}), u)$

(*b*) $\neg\text{Last}(y, z) \vee \text{Last}(\text{cons}(x, y), z)$

We then negate the goal to get:

(*c*) $\neg\text{Last}(\text{cons}(2, \text{cons}(1, \text{Nil})), v)$

Resolving (b) and (c), with m.g.u. {2/x cons(1, Nil/y, v/z), we get

(*d*) $\neg\text{Last}(\text{cons}(1, \text{Nil}), v)$

We can now resolve (a) and (b) with m.g.u. {1/e, 1/v} to get the empty clause.

2. The axioms (*a*) and (*b*) are the same as in part (1). In order to do answer extraction a predicate Ans (*v*) is appended to the clause containing the negated goal:

(*e*) $\neg\text{Last}(\text{cons}(2, \text{cons}(1, \text{Nil})), v) \vee \text{Ans}(v)$

Resolving (b) and (e), with m.g.u {2/x, con(1, Nil)/y, v/z}; we get:

(*f*) $\neg\text{Last}(\text{cons}(1, \text{Nil}), v) \vee \text{Ans}(v)$

We can now resolve (a) and (f) with m.g.u. {1/u, 1/v} to get:

(g) Ans (1)

Therefore the last element of the list (2, 1) is 1.

Q. 20. A logic circuit has four wires W1, W2, W3 and W4. It has an "and gate". A, and an "inverter", B. The input wires, W1 and W2, can be either "on" or not. If the and gate, A, is functioning properly (OK), wire W3 is "on" if and only if wires W1 and W2 both are "on". If the inverter, B, is functioning properly (OK), wire W4 is "on" if and only if wire W3 is not "on". Translate it into a set of WFFs.

Ans. $\{P(f_1(y)), P(f_2(y))\}, \{P(f_1(y)), Q(y, f_3(y))\}$

$\{\neg Q(f_1(y), y), P(f_2(y))\}, \{\neg Q(f_1(y), y), Q(y, f_3(y))\}$

$\{P(f_1(y)), P(f_2(y_1))\}, \{P(f_1(y_2)), Q(y, f_3(y_2))\}$

$\{\neg Q(f_1(y_3), y_3), P(f_2(y_3))\}, \{\neg Q(f_1(y_4), y_4), Q(y_4, f_3(y_4))\}$

Q. 21. **We are given following paragraph**

Tony, Mike and John belong to the Alpine Club. Every member of the Alpine Club is either a skier or a mountain climber or both. No mountain climber likes rain, and all skiers like snow. Mike dislikes whatever Tony likes and likes whatever Tony dislikes. Tony likes rain and snow.

Represent this information by predicate-calculus sentences in such a way that you can represent the question "Who is a member of the Alpine Club who is a mountain climber but not a skier?" as a predicate-calculus expression. Use resolution refutation with answer extraction to answer it.

Ans. One translation is the set of formulas

Member(Tony, Alpineclub)

Member(Mike, Alpineclub)

Member(John, Alpneclub)

$(\forall x)[\text{Member}(x\text{Alpineclub}) \supset (\text{Skier}(x) \vee \text{MC}(x))] \neg (\exists x)[\text{MC}(x) \wedge \text{Likes}(x, \text{Rain})]$

$(\forall x)[\text{Skier}(x) \supset \text{Likes}(x, \text{Snow})\}$

$(\forall x)[\text{Skier}(\text{Tony}, x) \wedge \neg \text{Likes}(\text{Mike}, x)\}$

$\text{Likes}(\text{Tony}, \text{Rain}) \wedge \text{Likes}(\text{Tony}, \text{Snow})$

The question is:

$(\exists x)\,[\text{Member}(x, \text{Alpineclub}) \wedge \text{MC}(x) \wedge \text{Skier}(x)]$

The clause forms of these sentences are:

1. Member(Tony, Alpineclub)
2. Member(Mike, Alpineclub)
3. Member(John, Alpineclub)
4. $\neg$Member(x, Alpineclub) $\vee$ Skier(x), MC(x)
5. $\neg$MC(x) $\vee$ $\neg$Likes(x, Rain)
6. $\neg$Skier(x) $\vee$ Likes(x, Snow)

7 *a*. $\neg$Likes(Tony, x) $\vee$ $\neg$Likes(Mike, x)

7 *b*. Likes(Tony, x) $\vee$ Likes(Mike, x)

8 *a*. Likes(Tony, Rain)

8 *b*. LIkes(Tony, Snow)

The clause form of the negation of the theorem to be proved (with Ans literal) is:

9. $\neg$Member(x, Alpineclub) $\vee$ $\neg$MC(x) $\vee$ Skier(x) $\vee$ Ans(x)

The refutation is:

10. ¬Member(x, Alpineclub), (Skier(x) ∨ Ans(x) resolving 9 with 4
11. Skier(Mike)/Ans(Mike) resolving 11 and 2
12. Likes(Mike, skier) ∨ ans (Mike) resolving 12 with 8
13. ¬Likes(Tony, Skiew) ∨ Ans(Mike) resolving 12 with 7a
14. Ans(Mike) resolving 13 with 8b.

Q. 22. Sam, Clyde and Oscar are elephants. We know the following facts about them:

1. Sam is pink.
2. Clyde is gray and likes Oscar.
3. Oscar is either pink or gray (but not both) and likes Sam.

Use resolution refutation to prove that a gray elephant likes pink elephant; that is, prove $(\exists x, y)[\text{Gray}(x) \wedge \text{Pink}(y) \wedge \text{Likes}(x, y)]$

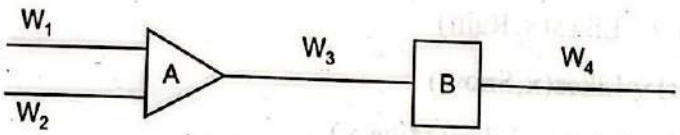
Ans. The clauses and negation of the theorem to be proved are:

1. P(Sam)
2. G(Clyde)
3. L(Clyde, Oscar)
4. P(Oscar) ∨ G(Oscar)
5. L(Oscar, Sam)
6. ¬G(x) ∨ ¬P(y) ∨ ¬L(x, y)

The resolution refutation is:

7. ¬G(Clyde) ∨ ¬P(Oscar) from 3 and 6
8. ¬P(Oscar) from 2 and 7
9. ¬G(Oscar) ∨ ¬P(Sam) from 5 and 6
10. ¬G(Oscar) from 1 and 9
11. P(Oscar) from 10 and 4
12. Nil from 11 and 8.

Q. 23. Consider the following circuit diagram —



1. Use expression like ON(A), ON(W1) and so on to describe the funtioning of the circuit as defined.
2. Using the formulas describing the functioning of the circuit and assuming that all components are functioning properly and that wires W1 and W2 are "on" use resolution to show that wire W4 is not "on".
3. Again, using the formulas describing the functioning of the circuit and given that wires W1 and W2 are "on" but that wire W4 is also "on", use resolution to show that either the and gate or the inverter is not functioning properly.

Ans. 1. (*a*) $OK(A) \supset [0n(W1) \wedge 0n(W)] \equiv 0n(W3)]$

(*b*) $OK(B) \supset [0n(W3) \equiv \neg 0n(W4)]$

2. The clause forms of the above formulas are:

(a1)$\neg OK(A) \vee \neg 0n(W2) \vee 0n(W3)$

(a2)$\neg OK(A) \vee \neg 0n(W3) \vee 0n(W1)$

(a3)$\neg OK(A) \vee \neg 0n(W3) \vee 0n(W2)$

(b1)$\neg OK(B) \vee \neg 0n(W3) \vee \neg 0n(W4)$

(b2)$\neg OK(B) \vee \neg 0n(W4) \vee \neg 0n(W3)$

Here is a resolution refutation showing that W4 is not "on"

1. 0n(W4) negation of theorem
2. $\neg OK(B) \vee \neg 0n(W3)$ resolving 1 with b1
3. $\neg 0n(W3)$ resolving 2 with hypothesis, OK(B)
4. $\neg OK(A) \vee \neg 0n(W1) \vee \neg 0n(W2)$ resolving 3 with a1
5. Nil successively resolving 4 with hypotheses, OK(A), 0n(W1), 0n(W2)

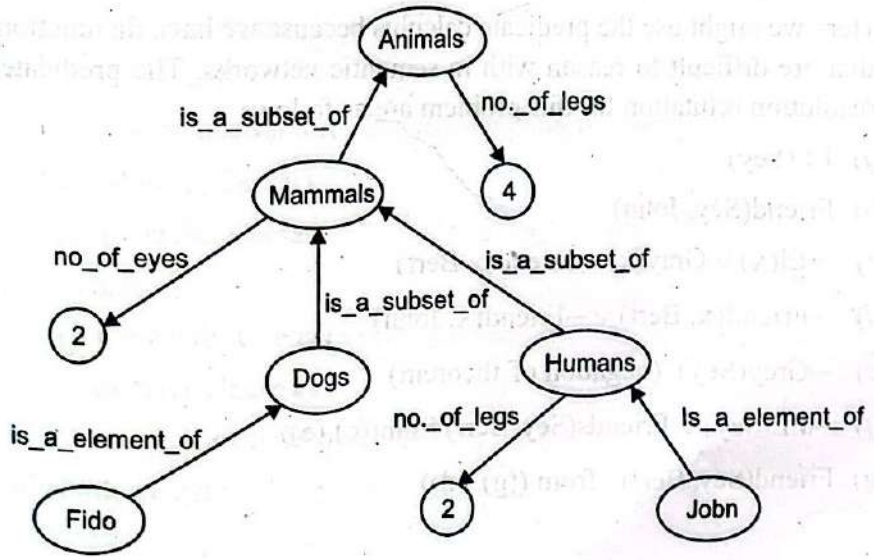
3. (1a) OK(A) negation of theorem

(1b) OK(B) negation of theorem

2. $\neg 0n(W1) \vee \neg 0n(W2) \vee 0n(W3)$ resolving 1a with a1
3. 0n(W3) successively resolving 2 with hypotheses, 0n(W1) and 0n(W2)
4. $\neg OK(B) \vee \neg 0n(W4)$ resolving 3 with b1
5. $\neg OK(B)$ resolving 4 with hypothesis, 0n(W4)
6. Nil resolving 5 with 1b.

Q. 24. 1. Express as predicate-calculus wffs the information in the semantic network shown here. Take care that the wffs are written in such a way that they explicitly state the inheritance cancellation information that is implicit in the inheritance hierarchy.

2. Can you prove that Fido has four legs from your predicate-calculus wffs? If not, what information would you need to add in order to complete the proof?



Ans. 1. The equivalent wffs are:

Def(Fido)

Human(John)

$(\forall x)[Dog(x) \supset Mammal(x)]$

$(\forall x)[Mammal(x) \supset Anjaml(x)]$

$(\forall x)[Human(x) \supset Mammal(x)]$

$(\forall x)[[Mammal(x) \wedge \neg Human(x)] \supset Haslegs(x,4)]$

$(\forall x)[Human(x) \supset Haslegs(x,2)]$

$(\forall x)[Mammal(x) \supset Haslegs(x,2)]$

2. We cannot prove Haslegs(Fido, 4) from these wffs. We would have to add $\neg$ Human(Fido). Or we could add something more general, such as $(\forall x)[Dog(x) \supset \neg Human(x)]$.

Q. 25. For the two examples shown here, pick the most appropriate knowledge representation system and encode the statements of the examples in the selected representation. Use your representation and a reasoning system appropriate to it to answer the question in each example. Example in some detail how the reasoning system arrives at the answer- showing resolution refutations, for example, if a reasoning system based on resolution is used. Use diagrams if they are helpful in your explanations.

1. Typically, birds fly. Ostriches are birds. Oliver is an ostrich. Ostriches do not fly. Q. Does Oliver fly?

2. Seymore is an elephant and John's friend. All elephants are gray or a friend of Bert. No friend of Bert is a friend of John. Q. Is Seymore gray?

Ans. 1. Here because of the nonmonotonic character of the statements, we might use a semantic network with property inheritance and cancellation of inheritance. In this case, we would have the fact the birds fly associated with the BIRDS node and the fact that ostriches do not fly associated with the OSTRICHES node. The OSTRICHES node is a descendant of the BIRDS node, and therefore, the fact that ostriches do not fly cancels the fact the birds do. Since Oliver is an ostrich, we conclude from the net that Oliver does not fly. (Note that the net formalism would allow us to say, in addition, that Oliver does fly, which we might do it we had some special information that Oliver was some sort of mutant flying ostrich).

2. Here we might use the predicate calculus because we have disjunction and statements that are difficult to reason with in semantic networks. The predicated encoding and resolution refutation for this problem are as follows

(*a*) EI (Sey)

(*b*) Friend(Sey, John)

(*c*) $\neg$El(x) $\vee$ Gray(x) $\vee$ Friend(x Bert)

(*d*) $\neg$Friend(x, Bert) $\vee \neg$Friend(x, John)

(*e*) $\neg$Grey(Sey) (negation of theorem)

(*f*) $\neg$EL(Sey) $\vee$ Friends(Sey, Bert) from((c),(e))

(*g*) Friend(Sey, Bert) from ((g), (d))

(*h*) ¬Friend(Sey, John) from ((g), (d))

(*i*) Nil from ((*h*), (*b*))

So, we have proved that Seymore is gray.

Q. 26. What is a stereotypical knowledge?

Ans. A type of knowledge that scripts attempt to capture with the aim of allowing a computer to make similar inferences about incomplete stories (Schank and Abelson, 1977).

Q. 27. What are demons in AI? Name some demons used in conventional programming too.

Ans. Demons are the procedures which are activated at anytime during a program execution depending on the conditions evaluated in demon itself. For example. –If–added procedure (executed when new information is placed in the slot), **If-removed procedure** (executed when information is deleted from the slot) and **If-needed procedure** (executed when information is needed from the slot but the slot is empty.

Examples of demons used in the conventional programming are—

(*a*) Error detection

(*b*) Default commands.

(*c*) End-of-File detection (EOF).

Q. 28. Given are the axioms in clause form:

1. **Man (Marcus)**
2. **Pompeian (Marcus)**
3. **~Pompeian (a_1) v Roman (a_1)**
4. **Ruler (Caesear)**
5. **~Roman (a_2) v loyal (a_2 Caesar) v hate (a_2, Caesar)**
6. **Loyal (a_3, f1(a_3))**
7. **~Main (a_4) v ~ ruler (b_1) v ~ assassin (a_4, b_1) v loyal (a_4, b_1)**
8. **assassin (Marcus, Caesar)**

Give resolution proof of hate(Marcus, Caesar).

Ans. To prove :-

Did Marcus hate Caesar?

We negate this statement as

~hate (Marcus, Caesar)

There is no axiom in clausal form starting with literal ~ hate.

Prove ~ hate (Marcus, Caesar)

Clause ~ hate (Marcus, Caesar)

Clause 5 has hate

~Roman (a_2) v loyal (a_2 Caesar) v hate (a_2 Caesar)

a_2 is Marcus we have clause as

~Roman (Marcus) v Loyal (Marcus, Caesar) v hate (Marcus Caesar)

Resolving both we get :-

~Roman (Marcus) v loyal (Marcus, Caesar)

Now select 3 clausal form.

~ Pompeian (a_1) v Roman (a_1) where a_1 is Marcus.

Statement will be

~ Pompeian (Marcus) V Roman (Marcus)

We get new clause after resolving.

C1: ~Roman (Marcus) v loyal (Marcus, Caesar)

C2: ~ Pompeian (Marcus) v Roman (Marcus)

Resolvent is ~ Pompeian (Marcus) v loyal (Marcus, Caesar)

Now take clause 2 which is

Pompeian (Marcus)

Resolvent is loyal (Marcus, Caesar)

Now C1 is loyal (Marcus, Caesar)

Select 7^{th} axiom as:

~man (a_4) v ~ ruler (b1) v ~ assassin (a_4, b1) V loyal (a_4, b_1)

where a_4 = Marcus

b_1 = Caesar

Statement will be

C2 ~ man (Marcus) v ~ ruler(Caesar) v ~ assassin (Marcus, Caesar) v loyal (Marcus Caesar)

Negate C1 as ~ loyal (Marcus, Caesar).

Resolvent will be:

~ man (Marcus) v ~ ruler (Caesar) v ~ assassin (Marcus, Caesar) V loyal (Marcus, Caesar)

Now clause is 1

man (Marcus)

Then new resolvent

~ ruler (Caesar) v ~ assassin (Marcus, Caesar) v loyal (Marcus, Caesar)

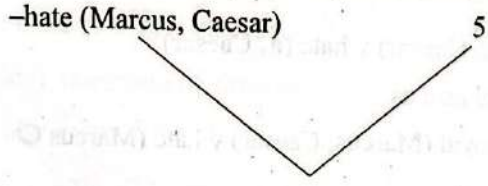
Clause 4 is ~ ruler (Caesar) v ~ assassin (Marcus, Caesar) v loyal (Marcus, Caesar) Resolvent is

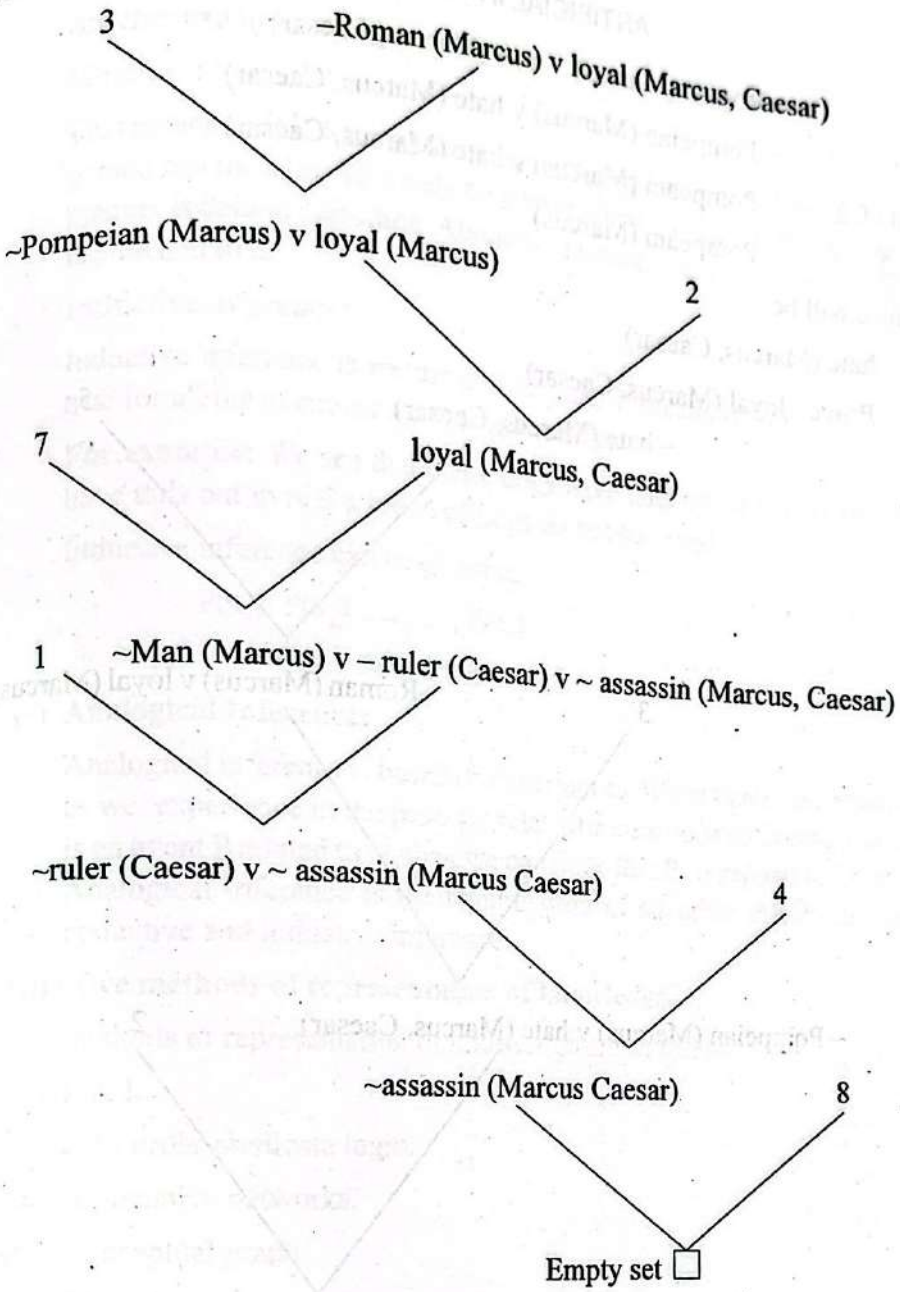
~ assassin (Marcus, Caesar)

Clause 8 is assassin (Marcus, Caesar)

Resolvent is empty

Diagramatically, it is shown as follows





Q. 29. Using above question

Prove : loyal (Marcus, Caesar)

Ans. Negate sentence as

~ loyal (M arcus, C aesar)

Select clause 5 which is

~Roman (x_2) v loyal (x_2 Caesar) v hate (x_2, Caesar)

where x_2 = Marcus

C1 : ~loyal (Marcus, Caesar)

C2 : ~ Roman (Marcus) v loyal (Marcus, Caesar) v hate (Marcus, Caesar)

Resolvent is ~Roman (Marcus) v hate (Marcus, Caesar) which is now

now C1.

Select 3 clausal from – ~ Pompeian (a) v Roman (a_1)

where a_1 is Marcus

C1 : ~Roman (Marcus) v hate (Marcus, Caesar)

C2 : ~ Pompeian (Marcus) v hate (Marcus, Caesar)

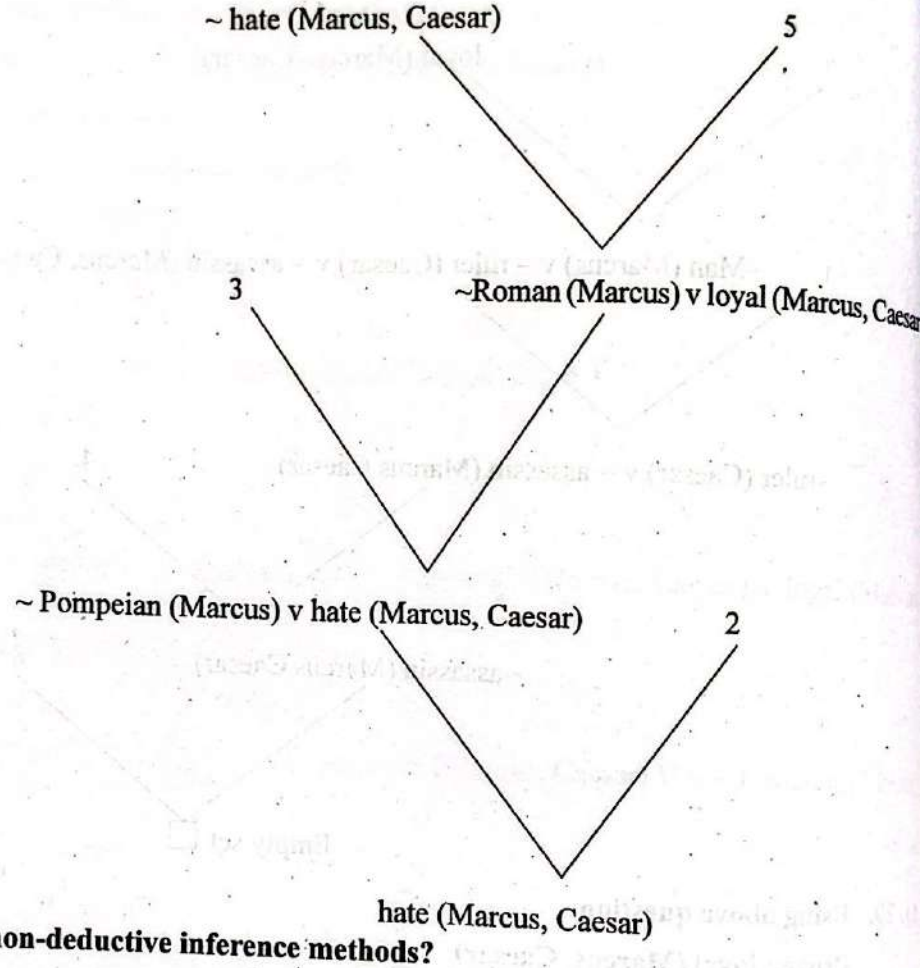
Next time C2 ~ Pompeain (Marcus) v hate (Marcus, Caesar)

Select C2 ~ Pompeain (Marcus)

New clause will be

hate (Marcus, Caesar)

Prove : loyal (Marcus, Caesar)



Q. 30. What are non-deductive inference methods?

Ans. Non-deductive inference: Non-deductive inferences are not valid forms of inferencing but are important.

There are three methods of non-deductive inference methods which are as follows:

(*a*) **Abductive inference:** It is based on the use of casual knowledge for describing conclusion.

Example

There are two axioms:

$\forall x : P(x) \rightarrow Q(x)$

P (R)

We can conclude that

Q (R)

Another example is

Measles (x) → Spots

This means people having measles get spots. If we notice the spots, we may conclude, person has measles. This may be a wrong conclusion. Deriving conclusions in such a manner is default reasoning. Abductive reasoning is useful if some measure of certainty is attached to it.

(*b*) **Inductive inference:**

Inductive inference is recurring inference. It means if there is some event, it will be true for all the events or entities in the class.

For example: We see that some dogs have tails by which we conclude that all dogs have tails but in real a breed of dogs do not have tail.

Inductive inference can be given as

$$P(a_1), P(a_2) \ldots\ldots, P(a_n)$$

$$\forall x : P(x)$$

(*c*) **Analogical Inference:**

Analogical inference is based on experiences. When there are certain similar situations as we experience in the past, we relate the situations or events. For example : If there is an event P related to Q, then we conclude that P^1 is related to Q^1 in the same manner. Analogical inference is the combination of all other inferences such as deductive, abductive and inductive inference.

Q. 31. Give five methods of representation of knowledge.

Ans. The methods of representation of knowledge are as follows

(*i*) FOPL

First order predicate logic.

(*ii*) Associative networks.

(*iii*) Conceptual graph.

(*iv*) Fuzzy logic.

(*v*) Frames.

EXERCISE QUESTIONS

Q. 1. (*a*) Represent the following sentences in predicate logic—

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

From these, answer the question "Was Marcus loyal to Caesar"?

(*b*) What are semantic networks? How can semantic networks be converted to frames?

(*c*) Write a script for "Eating in a restaurant". **[RTM Nagpur BE (IT) Summer 2009]**

Q. 2. (*a*) Explain difference between procedural and declarative knowledge.

(*b*) Assume the following facts —

1. Steve only likes easy courses.
2. Science courses are hard.
3. All courses in commerce department are easy.
4. COM302 is a commerce course.

(*c*) Convert the following into clausal form —

1. Everyone is loyal to someone.
2. John likes all kinds of food.

(*d*) Write short notes on —

1. Forward v/s Backward Reasoning.
2. Control knowledge.
3. CD
4. Scripts.

[RTM Nagpur, BE (IT) 7th Sem., Winter 2009]

Q. 3. (*a*) Explain the difference between predicate logic and propositional logic.

(*b*) Represent the following using predicate calculus notation —

1. There is a red part in bin that has a circular shape.
2. Raghav is computer science student but not a pilot or a football player.
3. The owner of car also owns the boat.
4. All people need air.

[RTM Nagpur, BE (CSE)-7th Sem., Summer 1996]

Q. 4. Assume the following facts —

1. Prasad only likes easy courses.
2. AI courses are hard.
3. All courses in Data Mining department are easy.
4. DM108 is a Data Mining Course.

(*a*) Use Resolution to answer the question- "Which course would Prasad like?"

(*b*) Also design clausal form for above sentences in FOPL using proper Axioms, Functions, Predicates, Variables and Constants.

(*c*) Hence with respect to resolution, explain. "What is proof by refutation"?

[RTM Nagpur University, BE (CSE)- 7th Sem., Summer 2004]

Q. 5. Express the following things in predicate calculus—

(*a*) All p's are q's.

(*b*) Some p's are q's.

(*c*) John hit everyone who was near him.

(*d*) Nobody does anything without a reason.

[PTU, MCA 4th Sem., 2009]

Q. 6. (a) What do you mean by semantic nets?

(b) Define frames.

[PTU B. Tech. (CSE)-7th/8th Sem., 2009]

Q. 7. (a) Give the CD structure to parse the following sentence —
"John wanted Mary to go to the store".

(b) Represent the following sentences using symbolic logic:

1. A drunker is enemy of himself.
2. Father of John loves the mother of mary.
3. All students like a good teacher.
4. Fruits and vegetables are nutritions.
5. God helps those who help themselves.

(c) "Some medicines are dangerous only if taken in excessive amount". Translate the above sentence into formulas in predicate logic and then to clausal form.

(d) Describe semantic net and frames with suitable examples.

[UPTU., B. Tech. (CS)- 7th Sem., 2007]

Q. 8. (a) Explain the difference between the following — Semantic Nets and Frames.

(b) Represent the following sentence in FOPL, convert it into clausal form explaning each step —

"All engineering students who know programming will either get good placement or admission in some university for higher studies".

(c) Using the following problem, illustrate how resolution is used to derive new facts from already existing facts —

Given facts

(i) Steve likes computer courses.

(ii) Computer software can be software or Hardware courses.

Derive: Steve likes AI.

(d) Write short notes on — Unification procedure.

[DTU (DCE)-ME (CS) 5th Sem., 2005]

Q. 9. Draw the semantic network and write atomic formulae for the English sentence given below:

"Manju playing with a red ball".

[Cochin University of Science & Technology B. Tech (CSE)-7th Sem., Oct. 1999]

Q. 10. (a) Construct a semantic net representations of the following —

(i) Rajesh is Indian, Rajesh is a lecturer.

(ii) Vikram but his new video CD to his brother-in-law.

(b) Explain-Conceptual dependency.

[GGSIPU, B.Tech. (CSE)-8th Sem., May 2008]

Q. 11. (a) Explain why these two sentences cannot be unified—

1. $p(x) \vee (q(f(x)) \wedge r(y))$

2. $p(x) \vee (q(f(r)) \wedge r(s))$

(b) Construct a semantic net representation of the following:

"Gopal is Indian, Gopal is lecturer".

[GGSIPU, B.Tech (CSE)-8th Sem., May-June 2009

Q. 12. (*a*) Create a script about shopping in a super market.

(*b*) Write down a substitution which unifies the following sentences:

1. likes(homer, dineer(x)) $\wedge$ (today(A) $\rightarrow$ dineer(x))
2. likes(homer, dineer(cooked_by(y)) $\wedge$ (today(Z) $\rightarrow$ dineerCcooked – by(magie)))

(c) What is a slot in a frame? **[GGSIPU, B.Tech (CSE)-8th Sem., May-June 2009**

Q. 13. What are the merits/demerits of semantic net over semantic frames?

[GGSIPU, B.Tech (CSE)-8th Sem., May 2010

Q. 14. (a) Explain the rules to unify two predicates.

(b) Represent the following facts as predicates and convert them to clausal form, explaining the steps of conversion:

1. Any student who is intelligent or hardworking will pass the exam in good marks
2. Anyone who is passed with good marks gets a job.

(*c*) Assume that "Shiva is neither hardworking nor intelligent". Using resolution prov that Shiva does not get a job. **[GGSIPU, B. Tech. (CSE) 8th Sem., May 2010**

Q. 15. What kind of knowledge is represented by the semantic nets? Give an example of KI represented by semantic nets and explain how inferences are done in Semantic Nets?

[GGSIPU, B. Tech. (CSE) 8th Sem., May 2010

Q. 16. (*a*) Create a script for "appearing for an examination."

(*b*) A new operator $\oplus$, or exclusive-or may be defined by the following truth table—

P	Q	$P \oplus Q$
T	T	F
T	F	T
F	T	T
F	F	F

Create a propositional calculus expression using $\wedge, \vee$ and $>$ that is equivalent to $P \oplus Q$.

(*c*) Explain CD using an example. **[GGSIPU, B. Tech. (CSE) 8th Sem., 2011]**