Unit-5
Knowledge Representation

Knowledge

Knowledge is a theoretical or practical understanding of a subject or a domain. Knowledge is also the sum of what is currently known.

Knowledge consists of information that has been interpreted, categorized, applied, experienced and refined over time.

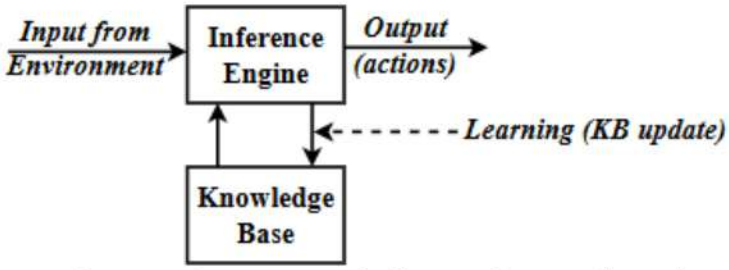
In general, knowledge is more than just raw data; it consists of facts, ideas, beliefs, heuristics, associations, rules, abstractions, relationships, and customs. It is used for reasoning, decision-making, and problem-solving within a specific domain.

Types of Knowledge

1. ***Procedural Knowledge:***
   - It is also known as imperative knowledge.
   - Procedural knowledge is a type of knowledge which gives information/knowledge about how to do achieve something.
   - It includes rules, strategies, procedures, agendas, etc.
   - *Example:* How to drive a car?
2. ***Declarative Knowledge:***
   - Declarative knowledge refers to facts or statements that describe the world, often in the form of "knowing what."
   - It is static and doesn't involve actions or procedures.
   - *Example:* "Kathmandu is the capital of Nepal" is a piece of declarative knowledge.
3. ***Structural Knowledge:***
   - Structural knowledge deals with the relationships between entities or concepts, organizing knowledge into structures like hierarchies or networks.
   - *Example:* How to various part of car fit together to make a car, or knowledge structures in terms of concepts, sub concepts and objects.
4. ***Meta Knowledge:***
   - Meta knowledge is knowledge about knowledge.
   - It refers to understanding which knowledge to use in a particular situation or context.
   - *Example:* In a medical diagnostic system, knowing which symptoms to prioritize when diagnosing a disease is an example of Meta knowledge.
5. ***Heuristic Knowledge:***
   - Heuristic knowledge indicates expert knowledge in any field or in any subject they have acquired over the years.
   - It can collect past problem experiences and, based on that, suggest a key approach for that problem and take action.
   - *Example:* A heuristic might be "If a website loads slowly, refresh the page" - it's not always guaranteed to work, but it's often useful.

Knowledge-Based Agent

A knowledge-based agent consists of a knowledge base (KB) and an inference engine (IE).



- A ***knowledge-base*** is a set of sentences of what one knows about the world.
- The ***inference engine*** derives new sentences from the input and KB.

The agent operates as follows:

1. It receives percepts from environment.
2. It computes what action it should perform (by IE and KB).
3. It performs the chosen action.

Knowledge Representation

Knowledge representation is the process of expressing the knowledge about the world in a computer tractable form. A formal language represents knowledge into a computer tractable form and the reasoning process manipulate it to deduce non- obvious facts.

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

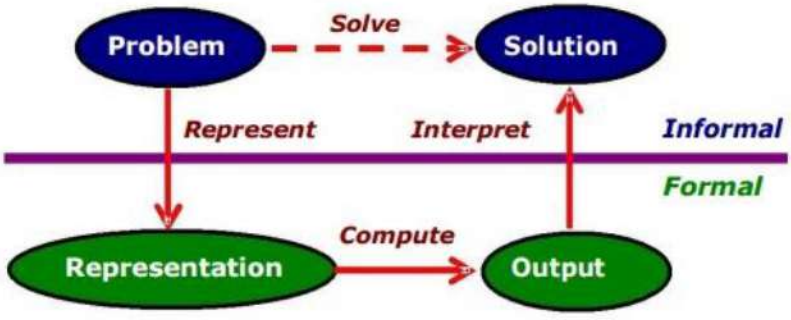


Fig. Knowledge Representation Framework

The steps are:

- The informal formalism of the problem takes place first.
- It is then represented formally and the computer produces an output.
- This output can then be represented in an informally described solution that user understands or checks for consistency.

Issues in Knowledge Representation

Several issues arise when using knowledge representation techniques. Some of these are:

- How can the problem be represented?
- What specific knowledge about the world is required?
- How can an agent acquire the knowledge from experts or from experience?
- How can the knowledge be debugged, maintained, and improved?
- Are there any attributes of object that occurs in almost every problem domain?
- Are there any important relationships between attributes of objects?
- At what level of detail should the knowledge be represented?
- How should sets of objects be represented?
- Given a large amount of knowledge stored, how can relevant parts be accessed when needed?

Knowledge Representation System Requirements

A knowledge representation system should have following ***properties***:

- ***Representational Adequacy:*** The ability to represent all kinds of knowledge that are needed in that domain.
- ***Inferential Adequacy:*** The ability to manipulate the represented structure and infer new structures.
- ***Inferential Efficiency:*** The ability to incorporate additional information into the knowledge structure that will aid the inference mechanisms.
- ***Acquisitional Efficiency:*** The ability to acquire new knowledge using automatic methods wherever possible rather than reliance on human intervention.

Knowledge Representation Techniques

There are several knowledge representation techniques in AI, including *logical representation*, *semantic network representation*, *frame representation*, and *production rules*.

Semantic Network

A semantic network is a graphical knowledge representation technique. Knowledge is represented as a collection of concepts, represented by nodes. Thus, semantic networks are constructed using ***nodes*** linked by directional lines called ***arcs.***

A node can represent a fact description such as
- physical object
- concept
- event

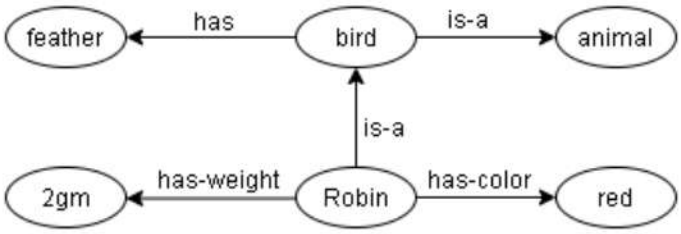
An arc (or link) represents relationships between nodes. There are some 'standard' relationship types:
- 'Is-a' (instance relationship): represent class/instance relationships
- 'Has-a' (part-subpart relationship): identify property relationships

Advantages of Semantic Network	*Disadvantages of Semantic Network*
▪ Easy to visualize and understand. ▪ Allows reasoning through inheritance and relationships. ▪ Reduces redundancy by representing shared properties at higher levels. ▪ The objects are represented only once.	▪ There is no standard definition for link names. ▪ It is unable to represent negation, quantification, disjunction etc. ▪ Complex relationships can lead to confusion without proper labeling.

Example:

All birds are animal.
Birds have feathers.
Robin is a bird.
Robin is red in color.
Weight of Robin is 2 gm.

The semantic network representation of given sentences is:

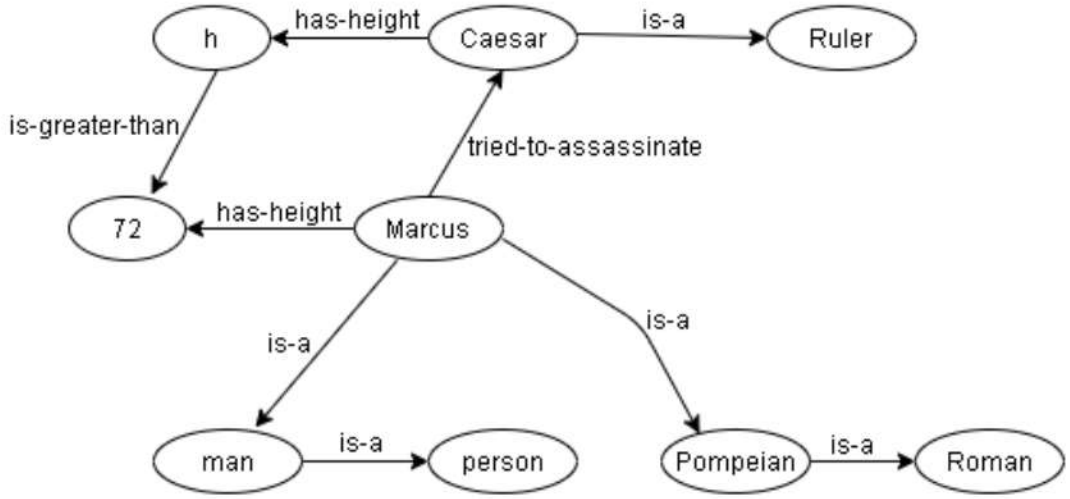


Q. Represent the following sentences into the semantic network.

All men are person.
All Pompeians are Roman.
Marcus, is a men, who is Pompeian.
Caesar is a ruler.
Marcus tried to assassinate Caesar.
Marcus has height 72.
Height of Marcus is less than the height of Caesar.

Solution:

The semantic network representation of given sentences is:

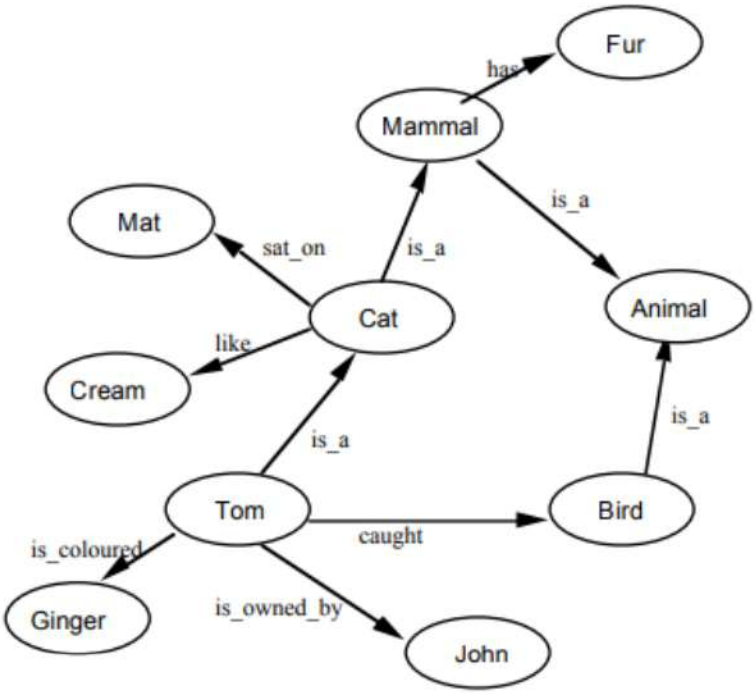


Q. Show the following statements in semantic network.

1. ***Tom is a cat.***
2. ***Tom caught a bird.***
3. ***Tom is owned by John.***
4. ***Tom is ginger in color.***

5. ***Cats like cream.***
6. ***The cat sat on the mat.***
7. ***A cat is a mammal.***
8. ***A bird is an animal.***
9. ***All mammals are animals.***
10. ***Mammals have fur.***

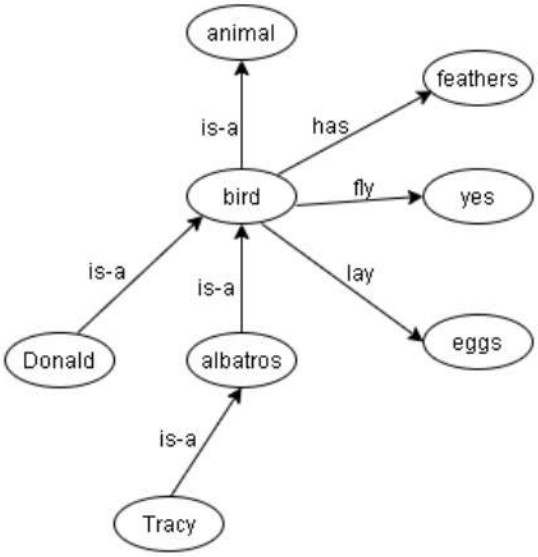
Solution:



Q. Draw a semantic network for the following facts:

Birds are animals.
Birds have feathers, fly and lay eggs.
Albators is a bird.
Donald is a bird.
Tracy is an albatross.

Solution:



Frames

A frame is a record like structure which consists of a collection of attributes and its values to describe an entity in the world. In this techniques, the knowledge is stored via slots and fillers.

Each frame has a name, slots (attributes) and fillers (values). ***Slots*** represent the attributes or properties of an entity. Each attribute or slot has a ***value*** attached to it. A particular value may be:

- a default value
- an inherited value from a higher frame
- a procedure, called a daemon, to find a value
- a specific value, which might represent an exception.

Example: A frame for a book is given below.

Book	
Slot	***Filler***
Name	Artificial Intelligence
Author	Peter Norvig
Year	2024
Price	Rs. 950

A single frame is not much useful. Frame systems usually have collection of frames connected to each other. Value of an attribute of one frame may be another frame.

The following figure shows the two frames: *Person* and *Address* connected with a link.

Person	
Name	Kushal
Profession	Doctor
Age	27
Wife	Nikita
Children	(Krish, Kriti)
Address	

Address	
Street	Thamel
City	Kathmandu
Country	Nepal

Types of Frame

- ***Class Frame:*** It represents generic concepts from which other frames inherit.
- ***Subclass Frame:*** It is derived from a class frame.
- ***Instance Frame:*** This represents a specific instance of a concept that cannot be further subclassed.

Types of Relationship

- ***Generalization:*** It denotes ***'a-kind-of'*** or ***'is-a'*** relationship between a superclass and its subclasses. Each subclass inherits all features of the superclass. For example, a car ***is a*** vehicle.
- ***Aggregation:*** It is ***'a-part-of'*** or ***'part-whole'*** relationship in which several subclasses representing components are associated with a superclass representing a whole. For example, an engine is ***a part of*** a car.
- ***Association:*** It describes some ***semantic relationship*** between different classes which are unrelated otherwise. For example, Mr. Jayanta owns a car and a computer. Such classes as Car and Computer are mutually independent, but they are linked with the frame Mr. Jayanta through the semantic association.

Advantages of Using Frames	Disadvantages of Using Frames
▪ Frames provide a clear and organized way to represent complex information. ▪ The inheritance mechanism allows for the reuse of frames, reducing redundancy. ▪ Frames can be easily updated or modified as new information becomes available. ▪ Frames make programming easier by grouping related knowledge together.	▪ As the number of frames and their interrelationships increase, managing and maintaining the frames can become complex. ▪ No standard for slots filler values. ▪ No associated reasoning or inference mechanism.

Rule Based Knowledge Representation System

A rule-based system (***or production system***) is a knowledge based system in which the knowledge is stored as production rules, sometimes called IF-THEN rules. A production system provides the mechanism necessary to execute production rules in order to achieve some goal for the system.

A production rules consist of two parts: The IF part, called antecedent (premise or condition) and the THEN part called the consequent (conclusion or action). The syntax structure is:

IF <condition> THEN <action>

For example, IF it is raining THEN open the umbrella.

A typical rule-based system has four basic components:

1. ***Working Memory:*** It contains facts about the world and can be observed directly or derived from a rule. It contains temporary knowledge – knowledge about this problem-solving session. It may be modified by the rules.
2. ***Rule Base:*** It contains rules; each rule is a step in a problem solving process. The syntax is a ***IF <conditions> THEN <actions>*** format. The conditions are matched to the working memory, and if they are fulfilled, the rule may be fired.
3. ***Interpreter:*** The processing engine which carries out reasoning on the rules and derives an answer. The interpreter operates on cycle:
   - ***Retrieval:*** Finds the rules that match the current Working Memory. These rules are the Conflict Set.
   - ***Refinement:*** One of the rule in the conflict set is then selected.
   - ***Execution:*** Apply the rule.
4. ***A user interface*** or other connection to the outside world through which input and output signals are received and sent.

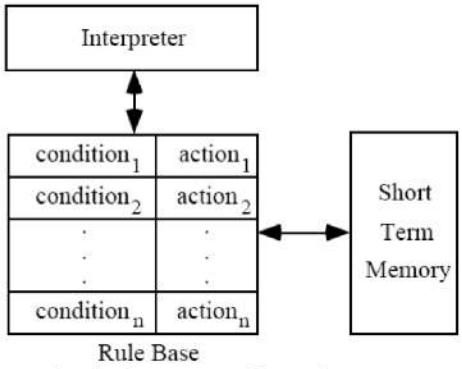


Fig: Block diagram of production system

Logic Based Knowledge Representation

Logical representation uses formal logic to encode knowledge, allowing AI to reason by applying rules and deriving conclusions. Logic makes statements about the world which are true (or false).

Each sentence could be translated into logics using the *syntax* and *semantics*.

- ***Syntax*** are the rules which decide how we construct legal sentences in the logic. ***(Symbols)***
- ***Semantics*** are the rules by which we can interpret the sentence in the logic. (***Meaning of sentence***).

There are two types of logical representation:

1. Propositional Logic and
2. First-order Logic.

Propositional Logic

Propositional Logic (PL) is the simplest form of logic where all the statements are made by propositions. A proposition is a declarative sentence which is either true or false, but it cannot be both. It is a technique of knowledge representation in logic and mathematical form. In propositional logic, we use symbolic variables to represent the logic.

Formal grammar/syntax for propositional logic can be given as below:

$$\begin{aligned} Sentence &\rightarrow AtomicSentence \mid ComplexSentence \\ AtomicSentence &\rightarrow \text{True} \mid \text{False} \mid Symbol \\ Symbol &\rightarrow \text{P} \mid \text{Q} \mid \text{R} \ldots \\ ComplexSentence &\rightarrow \neg Sentence \\ &\mid (Sentence \wedge Sentence) \\ &\mid (Sentence \vee Sentence) \\ &\mid (Sentence \Rightarrow Sentence) \\ &\mid (Sentence \Leftrightarrow Sentence) \end{aligned}$$

Logical Connectives

Logical connectives are used to connect two simpler propositions or representing a sentence logically. We can create compound propositions with the help of logical connectives. There are mainly five connectives which are given as follows:

- $Negation(\neg)$,
- $Conjunction(\wedge)$,
- $Disjunction(\vee)$,
- $Implication(\Rightarrow)$,
- $Biconditional\ (\Leftrightarrow)$

Truth Table in Propositional Logic

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
False	False	True	False	False	True	True
False	True	True	False	True	True	False
True	False	False	False	True	False	False
True	True	False	True	True	True	True

Well-Formed Formula

Well-formed formula (sentence) is defined as:

- An atomic proposition S is a well-formed formula.
- If S is a well-formed formula, then $\neg S$ is a well-formed formula.
- If S and T are well-formed formulas, then $(S \wedge T)$, $(S \vee T)$, $(S \Rightarrow T)$ and $S \Leftrightarrow T$ are also well-formed formulas.
- A propositional expression is a well-formed formula if and only if it can be obtained by using above conditions.

Some Terminologies

- ***Tautology (Validity):*** A proposition that is always true, no matter what the truth values of the propositional variables that contain in it, is called a *tautology*.
- ***Contradiction (Unsatisfiable):*** A proposition that is always false, no matter what the truth values of the propositional variables that contain in it, is called a *contradiction.*
- ***Contingency (Satisfiable):*** A proposition that is neither a tautology nor a contradiction is called a *contingency*.

Q. State whether the following sentences are valid, unsatisfiable, or neither.

a. $S \Rightarrow S$
b. $S \Rightarrow F$
c. $S \wedge \neg S$

Solution:

a. $S \Rightarrow S$

S	**S**	**$S \Rightarrow S$**
T	T	T
F	F	T

Here, $(S \Rightarrow S)$ is true false for all interpretations. Hence it is a valid sentence.

b. $S \Rightarrow F$

S	**F**	**$S \Rightarrow F$**
T	T	T
T	F	F
F	T	T
F	F	T

It is neither valid nor unsatisfiable i.e. it is satisfiable.

c. $S \wedge \neg S$

S	**$\neg S$**	**$S \wedge \neg S$**
T	F	F
F	T	F

Here, the given sentence is false for all interpretations. Hence, it is unsatisfiable.

Inference Rules in Propositional Logic

Rule of Inference	*Name*	*Rule of Inference*	*Name*
$\begin{array}{l} p \\ p \rightarrow q \\ \hline \therefore q \end{array}$	Modus ponens	$\begin{array}{l} \neg q \\ p \rightarrow q \\ \hline \therefore \neg p \end{array}$	Modus tollens
$\begin{array}{l} p \rightarrow q \\ q \rightarrow r \\ \hline \therefore p \rightarrow r \end{array}$	Hypothetical syllogism	$\begin{array}{l} p \vee q \\ \neg p \\ \hline \therefore q \end{array}$	Disjunctive syllogism
$\begin{array}{l} p \\ \hline \therefore p \vee q \end{array}$	Addition	$\begin{array}{l} p \wedge q \\ \hline \therefore p \end{array}$	Simplification
$\begin{array}{l} p \\ q \\ \hline \therefore p \wedge q \end{array}$	Conjunction	$\begin{array}{l} p \vee q \\ \neg p \vee r \\ \hline \therefore q \vee r \end{array}$	Resolution

Q. "We will be happy. If we will be happy then sun will shine. The sun will shine if and only if either it will rain or it will not be cloudy. It will be cloudy but I will not be happy". Represent the above sentence in propositional logic.

Solution:

Let,

p $\rightarrow$ We will be happy.
q $\rightarrow$ Sun will shine.
r $\rightarrow$ It will rain.
s $\rightarrow$ It will be cloudy.

Now, converting into proposition,

1. p
2. $p \rightarrow q$
3. $q \leftrightarrow (r \vee \neg s)$
4. $s \wedge \neg p$

Conjunctive Normal Form (CNF)

A sentence that is expressed as a conjunction of disjunction of literals is said to be in conjunctive normal form (CNF). For example: $(P \vee Q) \wedge (P \vee R)$

CNF conversion steps:

1. Eliminate $\leftrightarrow$ rewriting $P \leftrightarrow Q$ as $(P \rightarrow Q) \wedge (Q \rightarrow P)$
2. Eliminate $\rightarrow$ rewriting $P \rightarrow Q$ as $\neg P \vee Q$
3. Use De Morgan's laws to push $\neg$ inwards:
   - rewrite $\neg(P \wedge Q)$ as $\neg P \vee \neg Q$
   - rewrite $\neg(P \vee Q)$ as $\neg P \wedge \neg Q$
4. Eliminate double negations: rewrite $\neg\neg P$ as P
5. Use the distributive laws to get CNF:
   - rewrite $P \vee (Q \wedge R)$ as $(P \vee Q) \wedge (P \vee R)$

6. Use:
   - $(P \wedge Q) \wedge R$ as $P \wedge Q \wedge R$
   - $(P \vee Q) \vee R$ as $P \vee Q \vee R$

Example: Consider the knowledge base given as below:

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

Converting to CNF:

$$\begin{aligned}
&(P \rightarrow Q) \rightarrow R \\
&\equiv (\neg P \vee Q) \rightarrow R \\
&\equiv \neg(\neg P \vee Q) \vee R \\
&\equiv (P \wedge \neg Q) \vee R \\
&\equiv \underbrace{(\boldsymbol{P \vee R}) \wedge (\boldsymbol{\neg Q \vee R})}_{\text{CNF}}
\end{aligned}$$

Q. Convert the given fact into CNF.

$$\boldsymbol{b \leftrightarrow (a \vee c)}$$

Solution:

$$\begin{aligned}
&b \leftrightarrow (a \vee c) \\
&(b \rightarrow (a \vee c)) \wedge ((a \vee c) \rightarrow b) \\
&(\neg b \vee (a \vee c)) \wedge (\neg (a \vee c) \vee b) \\
&(\neg b \vee a \vee c) \wedge (\neg (a \vee c) \vee b) \\
&(\neg b \vee a \vee c) \wedge ((\neg a \wedge \neg c) \vee b) \\
&(\neg b \vee a \vee c) \wedge ((\neg a \vee b) \wedge (\neg c \vee b)) \\
&(\neg b \vee a \vee c) \wedge (\neg a \vee b) \wedge (\neg c \vee b)
\end{aligned}$$

Now Knowledge Base (KB) contains following sentences in CNF:

1. $(\neg b \vee a \vee c)$
2. $(\neg a \vee b)$
3. $(\neg c \vee b)$

Disjunctive Normal Form (DNF)

A sentence that is expressed as a disjunction of conjunction of literals is said to be in Disjunctive Normal Form (DNF). For example, $(P \wedge Q) \vee (P \wedge R)$

Converting to DNF:

The steps for converting a formula to ***Disjunctive Normal Form (DNF)*** are almost identical to those for converting to ***Conjunctive Normal Form (CNF)***. The key difference lies in how the distribution is handled in Step 5.

- Instead of distributing ***∨ (OR)*** over ***∧ (AND)***, in DNF conversion, you apply the distribution of ***∧ (AND)*** over ***∨ (OR)***.

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

Resolution

Resolution is a valid inference rule producing a new clause implied by two clauses containing complementary literals.

- ***Unit resolution rule:*** Unit resolution rule takes a disjunction of literals and a literal and produces a new clause.

$$\frac{P_1 \vee P_2 \vee \ldots\ldots \vee P_{i-1} \vee P_i \vee P_{i+1} \vee \ldots\ldots \vee P_n, \quad q}{P_1 \vee P_2 \vee \ldots\ldots \vee P_{i-1} \vee P_{i+1} \vee \ldots\ldots \vee P_n}$$

Where P_i and q are complementary literals.

E.g.

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

- ***Generalized resolution rule:*** Generalized resolution rule takes two clauses of any length and produce a new clause.

$$\frac{p_1 \vee p_2 \vee \ldots \vee p_{i-1} \vee p_i \vee p_{i+1} \vee \ldots \vee p_n, q_1 \vee q_2 \vee \ldots \vee q_{j-1} \vee q_j \vee q_{j+1} \vee \ldots \vee q_m}{p_1 \vee p_2 \vee \ldots \vee p_{i-1} \vee p_{i+1} \vee \ldots \vee p_n \vee q_1 \vee q_2 \vee \ldots \vee q_{j-1} \vee q_{j+1} \vee \ldots \vee q_m}$$

Where p_i and q_j are complementary literals.

E.g.

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

Resolution Algorithm to Infer Conclusion

We can use the resolution algorithm to infer conclusion as follows:

1. Negate the query to be proven (proof by contradiction).
2. Convert Knowledge Base (KB) and negated query into CNF and extract clauses.
3. Repeatedly apply resolution rule to clauses until either the empty clause (contradiction) is derived or no more clauses can be derived.
4. If the empty clause is derived, answer 'yes' (query follows from knowledge base), otherwise answer 'no' (query does not follow from knowledge base)

Q. Given fact

$(\boldsymbol{b} \leftrightarrow (\boldsymbol{a} \vee \boldsymbol{c})) \wedge \neg \boldsymbol{b}$

Infer a.

Solution:

Convert KB into CNF

$(b \leftrightarrow (a \vee c)) \wedge \neg b$

$[(b \rightarrow (a \vee c)) \wedge ((a \vee c) \rightarrow b)] \wedge \neg b$

$[(\neg b \vee (a \vee c)) \wedge (\neg(a \vee c) \vee b)] \wedge \neg b$

$[(\neg b \vee a \vee c) \wedge ((\neg a \wedge \neg c) \vee b)] \wedge \neg b$

$(\neg b \vee a \vee c) \wedge (\neg a \vee b) \wedge (\neg c \vee b) \wedge \neg b$

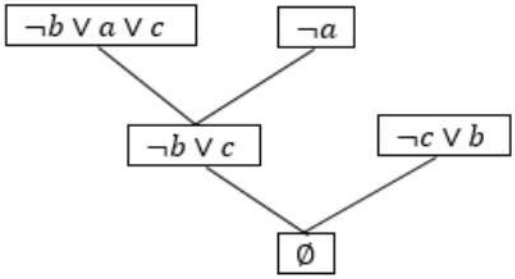
Now add negation of sentence to be inferred into KB.

Then KB contains following sentences in CNF:

1. $(\neg b \vee a \vee c)$
2. $(\neg a \vee b)$

3. $(\neg c \vee b)$
4. $\neg b$
5. $\neg a$ (negation of conclusion)

Now use Resolution algorithm:



Since an empty clause (∅) has been deduced through the resolution process, we can conclude that 'a' can be inferred from the given fact.

Q. "If it is hot and humid, then it is raining. If it is humid, then it is hot. It is humid". Is it raining?

Solution:

Let,

p → It is hot.
q → It is humid.
r → It is raining.

The KB contains following facts:

1. $(p \wedge q) \rightarrow r$
2. $q \rightarrow p$
3. q

We aim to show r. So add it's negation into KB.

4. $\neg r$

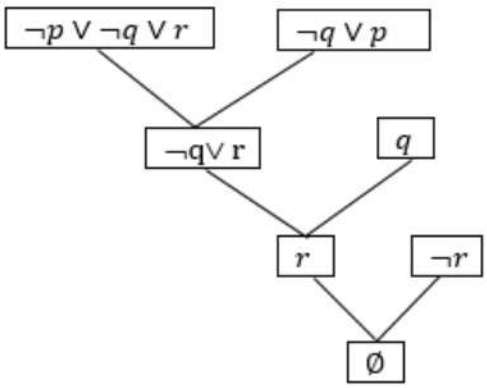
Converting KB into CNF

1. $(p \wedge q) \rightarrow r \equiv \neg(p \wedge q) \vee r \equiv \neg p \vee \neg q \vee r$
2. $q \rightarrow p \equiv \neg q \vee p$
3. q
4. $\neg r$

Now, the KB in CNF:

1. $\neg p \vee \neg q \vee r$
2. $\neg q \vee p$
3. q
4. $\neg r$

Apply resolution:



Since an empty clause (∅) has been deduced through the resolution process, we can conclude that r (it is raining) is true.

First-Order Predicate Logic (FOPL)

In propositional logic (PL), we can only represent the facts, which are either true or false. PL is not sufficient to represent the complex sentences or natural language statements. The propositional logic has very limited expressive power. Consider the following sentence, which we cannot represent using PL logic.

- "Some humans are intelligent", or
- "Sandip likes cricket."

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

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

- ***Objects:*** A, B, people, house, numbers, colors, theories etc.
- ***Relations:*** red, round, is adjacent, brother of, bigger than etc.
- ***Function:*** Father of, best friend, third inning of, end of etc.

Syntax for first-order predicate logic (FOPL) can be given as below:

Sentence	→	*AtomicSentence*
	|	*(Sentence* `Connective` *Sentence)*
	|	`Quantifier` *Variable, ... Sentence*
	|	¬ *Sentence*
AtomicSentence	→	*Predicate(Term, ...)* | *Term = Term*
Term	→	*Function(Term, ...)* | *Constant* | *Variable*
Connective	→	∧ | ∨ | ⇒ | ⇔
Quantifier	→	∀ | ∃
Constant	→	*A, B, C,* X_1, X_2, *Jim, Jack*
Variable	→	*a, b, c,* x_1, x_2, *counter, position, ...*
Predicate	→	*Adjacent-To, Younger-Than, HasColor, ...*
Function	→	*Father-Of, Square-Position, Sqrt, Cosine*

Quantifier

There is need to express properties of entire collections of objects, instead of enumerating the objects by name. Quantifiers let us do this.

FOL contains two standard quantifiers called

a) Universal: 'for all' (∀)
b) Existential: 'there exists' (∃)

- ***Universal Quantification:*** It makes statements about every object.
  $\forall x\ P(x)$: means that P holds for **all** values of x in the domain associated with that variable.
  E.g. $\forall x\ dolphin(x) \Rightarrow mammal(x)$
- ***Existential Quantification:*** It makes a statement about some object in the universe without naming it.
  $\exists x\ P(x)$: means that P holds for **some** value of x in the domain associated with that variable.
  E.g. $\exists x\ mammal(x) \wedge lays\_eggs(x)$

> ***Note:*** The main connective for universal quantifier ∀ is implication →
> The main connective for existential quantifier ∃ is implication ∧

Universal Instantiation and Existential Instantiation (Skolemization)

- ***Universal Instantiation (UI):*** This allows universal quantifiers to be removed. It substitutes ground term (term without variables) for the variables. For example:
  $\forall x\ Pompeian(x) \Rightarrow Roman(x)$,
  $Pompeian(Marcus)$
  $Roman(Caser)$

  Its UI is:
  $Pompeian(Marcus) \Rightarrow Roman(Caser)$
  $Pompeian(Marcus)$
  $Roman(Caser)$

- ***Existential Instantiation (EI) / Skolemization:*** This allows for existential quantifiers to be removed. It is the process of replacing existential quantified variable with constant that is not in KB (i.e. skolem constant) and deletion of the respective quantifier. For example:
  $\exists x\ mammal(x) \wedge lays\_eggs(x)$

  After EI,
  $mammal(C1) \wedge lays\_eggs(C1)$ where $C1$ is skolem constant.

FOPL Conversion Examples

1. ***Marcus was a man.***

   Let, Man$(x) \rightarrow x$ is a man.

   The fact in FOPL is:
   $Man(Marcus)$

2. ***Marcus was a Pompeian.***

   Let, Pompeian$(x) \rightarrow x$ is a Pompeian.

   The fact in FOPL is:
   $Pompeian(Marcus)$

3. ***All Pompeian's were roman.***

   Let, Pompeian$(x) \rightarrow x$ is a Pompeian.
   Roman$(x) \rightarrow x$ is a Roman.

   The fact in FOPL is:
   $\forall x\ (Pompeian(x) \rightarrow Roman(x))$

4. ***Ceasor was a ruler.***

   Let, Ruler$(x) \rightarrow x$ is a ruler.

   The fact in FOPL is:
   $Ruler(Ceasor)$

5. ***All romans were either loyat to ceasor or hated him.***

   Let, Roman$(x) \rightarrow x$ is a Roman.
   LoyalTo$(x, y) \rightarrow x$ is loyal to y.
   Hates$(x, y) \rightarrow x$ hates y.

   The fact in FOPL is: $\forall x\ (Roman(x) \rightarrow (LoyalTo(x, Caesar) \vee Hates(x, Caesar)))$

6. ***Everyone is loyal to someone.***

   Let, LoyalTo$(x, y) \rightarrow x$ is loyal to y.

   The fact in FOPL is:

   $\forall x \; \exists y \; LoyalTo(x, y)$

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

   Let, Person$(x) \rightarrow x$ is a person.
   Ruler$(x) \rightarrow x$ is a ruler.
   TriesToAsassinate$(x, y) \rightarrow x$ tries to assassinate y.
   LoyalTo$(x, y) \rightarrow x$ is loyal to y.

   The fact in FOPL is:

   $\forall x \; \forall y \; ((Person(x) \wedge Ruler(y) \wedge TriesToAsassinate(x, y)) \rightarrow \neg LoyalTo(x, y))$

Q. Translate the following sentences into first order logic:

i. "Everyone's DNA is unique and is derived from their parents' DNA".
ii. "No dog bites a child of its owner".
iii. "Every gardener likes the sun".
iv. "All purple mushrooms are poisonous".
v. "No two adjacent countries have the same color".
vi. "Politicians can fool some of the people all of the time".
vii. "There is a barber who shaves all men in town who do not shave themselves."

Solution:

i) Let's assume, for FOPL, the following predicates:

DNA(x) → x's DNA.
Unique(x, y) → x and y are unique.
Derived_from(x, y) → x is derived from y.
Parent(x, y) → x is parent of y.

The fact in FOPL is:

$$\forall x \; \forall y \; \forall z \; [Unique(DNA(x), DNA(y)) \rightarrow (DNA(x) \wedge DNA(y) \wedge Parent(y, x) \wedge Derived\_from(y, z) \wedge Parent(z, y))]$$

ii) Let's assume, for FOPL, the following predicates:

Dog(x) → x is a dog.
Child(x, y) → x is child of y.
Bites(x, y) → x bites y.
Owner(x, y) → x is owner of y.

The fact in FOPL is:

$$\forall x \; \forall y \; \forall z \; [(Dog(x) \wedge Owner(y, x) \wedge Child(z, y)) \rightarrow \neg Bites(x, z)]$$

iii) Let's assume, for FOPL, the following predicates:

Gardener(x) → x is gardener.
Likes(x, y) → x likes y.

The fact in FOPL is:

$$\forall x \; (Gardener(x) \rightarrow Likes(x, sun))$$

iv) Let's assume, for FOPL, the following predicates:

Purple(x) → x is purple.
Mushroom(x) → x is mushroom.
Poisonous(x) → x is poisonous.

The fact in FOPL is:

$$\forall x\ [(Mushroom(x) \wedge Purple(x)) \rightarrow Poisonous(x)]$$

v) Let's assume, for FOPL, the following predicates:

Country(x) → x is country.
Adj(x, y) → x and y are adjacent.
Distinct(x, y) → x and y are distinct.
Color(x) → x is color.

The fact in FOPL is:

$$\forall x\, \forall y\ [(Country(x) \wedge Country(y) \wedge Adj(x,y)) \rightarrow Distinct\ (Color(country(x)), Color(country(y))]$$

vi) Let's assume, for FOPL, the following predicates:

Person(x) → x is person.
Fool(x, y, t) → x can fool y at time t.

The facts in FOPL is:

$$\exists x\, \forall t\ [Person(x) \rightarrow Fools(Politicians, x, t)]$$

vii) Let's assume, for FOPL, the following predicates:

Barber(x) → x is a barber.
Man(x) → x is a man.
Shaves(x, y) → x shaves y.

The facts in FOPL is:

$$\exists x\, \forall y\ [(Barber(x) \wedge Man(y) \wedge \neg Shaves(y,y)) \rightarrow Shaves(x,y)]$$

Conversion to CNF

1. Eliminate implications and bi-implications.
   - Rewrite $P \leftrightarrow Q$ as $(P \rightarrow Q) \wedge (Q \rightarrow P)$
   - Rewrite $P \rightarrow Q$ as $\neg P \vee Q$
2. Move negations inward using De Morgan's laws.
   - rewrite $\neg(P \wedge Q)$ as $\neg P \vee \neg Q$
   - rewrite $\neg(P \vee Q)$ as $\neg P \wedge \neg Q$

   Plus rewriting $\neg\forall x P\ as\ \exists x \neg P$ and $\exists x P\ as\ \forall x \neg P$
3. Eliminate double negations.
   - rewrite $\neg\neg P$ as P
4. Rename bound variables if necessary so each only occurs once.
   - e.g. $\forall x P(x) \vee \exists x Q(x)$ becomes $\forall x P(x) \vee \exists y Q(y)$

5. Use equivalences to move quantifiers to the left.
   - e.g. $\forall x P(x) \wedge Q$ becomes $\forall x(P(x) \wedge Q)$ where x is not in Q
   - e.g. $\forall x P(x) \wedge \exists y Q(y)$ becomes $\forall x \exists y(P(x) \wedge Q(y))$

6. Skolemise (replace each existentially quantified variable by a new term)
   - $\exists x P(x)$ becomes $P(C1)$ using a Skolem constant $C1$ since $\exists x$ occurs at the outermost level.
   - $\forall x \exists y P(x, y)$ becomes $P(x, f_0(x))$ using a Skolem function f_0 since $\exists y$ occurs within $\forall x$.
7. The formula now has only universal quantifiers and all are at the left of the formula; drop them.
8. Use distribution laws to get CNF and then clausal form.
   - rewrite $P \vee (Q \wedge R)$ as $(P \vee Q) \wedge (P \vee R)$

Resolution in FOPL

- Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e., proofs by contradictions.
- Resolution is used, if there are various statements are given, and we need to prove a conclusion of those statements. ***Unification*** is a key concept in proofs by resolutions.
- Resolution is a single inference rule which can efficiently operate on the conjunctive normal form or clausal form**.**

Steps for Resolution:

1. Conversion of facts into first-order logic.
2. Negate the query to be proven (proof by contradiction).
3. Convert FOL statements into CNF and extract clauses.
4. Repeatedly apply resolution rule to clauses until either the empty clause (contradiction) is derived or no more clauses can be derived.
5. If the empty clause is derived, answer 'yes' (query follows from knowledge base), otherwise answer 'no' (query does not follow from knowledge base)

Unification

Unification is a process of making two different logical atomic expressions identical by finding a substitution. Unification depends on the substitution process. It takes two literals as input and makes them identical using substitution. For example:

Literal 1	**Literal 2**	**Unifier**
Knows(John, x)	Knows(John, Ram)	{x / Ram}
Knows(John, x)	Knows(y, Ram)	{x / Ram, y / John}
Knows(John, x)	Knows(y, mother(y))	{y/John, x/mother(y)}
Knows(John, x)	Knows(x, Ram)	{fail}

Last unification is failed due to overlap of variables, x cannot take the values of John and Ram at the same time.

Q. Assume the following facts:

John likes all kinds of food.
Apples are food.
Chicken is food.
Anything anyone eats and isn't killed by is food.
Bills eats peanuts and is still alive.
Sue eats everything Bill eats.

Prove that John likes peanuts using resolution.

Solution:

Let's assume, for FOPL, the following predicates:

Food(x) → x is food.
Likes(x, y) → x likes y.
Eats(x, y) → x eats y.
KilledBy(x, y) → x is killed by y.

The facts in FOPL are:

1. ∀x [Food(x) → Likes (John, x)]
2. Food (Apples)
3. Food (Chicken)
4. ∀x∀y [(Eats(x, y) ∧ ¬KilledBy(x, y)) → Food(y)]
5. Eats (Bill, Peanuts) ∧ ¬KilledBy (Bill, Peanuts)
6. ∀x [Eats(Bill, x) → Eats(Sue, x)]

Negating the statement which has to be proved:

7. ¬Likes(John, Peanuts) (*Negation of the statement to be proved*)

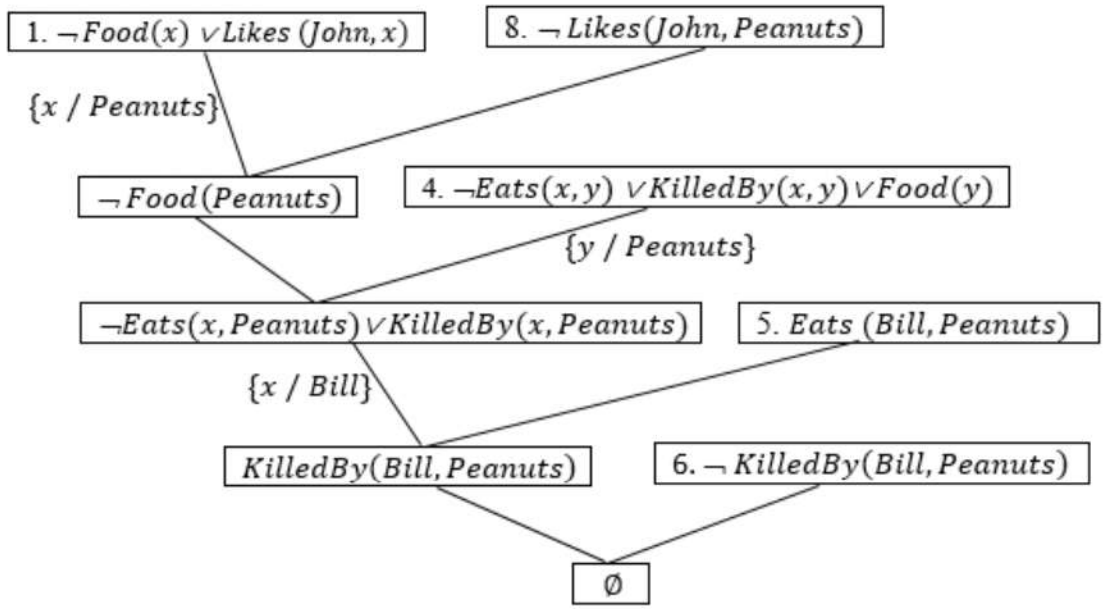
Converting to CNF,

1. ∀x [Food(x) → Likes (John, x)]
   ∀x [¬Food(x) ∨ Likes (John, x)]
   ¬Food(x) ∨ Likes (John, x)
2. Food (Apples)
3. Food (Chicken)
4. ∀x∀y[(Eats(x, y) ∧¬KilledBy(x, y)) → Food(y)]
   ∀x∀y[¬(Eats(x, y) ∧¬KilledBy(x, y)) ∨ Food(y)]
   ∀x∀y[(¬Eats(x, y) ∨ KilledBy(x, y)) ∨ Food(y)]
   ¬Eats(x, y) ∨ KilledBy(x, y)∨ Food(y)
5. Eats (Bill, Peanuts)
6. ¬KilledBy (Bill, Peanuts)
7. ∀x[Eats(Bill, x) → Eats(Sue, x)]
   ∀x[¬Eats(Bill, x) ∨ Eats(Sue, x)]
   ¬Eats(Bill, x) ∨ Eats(Sue, x)
8. ¬Likes(John, Peanuts)

The KB contains following sentences in CNF,

1. ¬Food(x) ∨ Likes (John, x)
2. Food (Apples)
3. Food (Chicken)
4. ¬Eats(x, y) ∨ KilledBy(x, y)∨ Food(y)
5. Eats (Bill, Peanuts)
6. ¬KilledBy (Bill, Peanuts)
7. ¬Eats(Bill, x) ∨ Eats(Sue, x)
8. ¬Likes(John, Peanuts)

Now, Using Resolution:



Since an empty clause has been deduced we say that our assumption is wrong and hence we have proved "John likes Penauts".

Q. Consider the following English statements

1. ***Everyone who are graduating are happy.***
2. ***Everyone who are happy smiles.***
3. ***Someone is graduating.***

Convert the above statements in FOPL and using resolution prove that "someone is smiling".

Solution:

Let's assume, for FOPL, the following predicates:

Graduating(x) → x is graduating.

Happy(x) → x is happy.

Smile(x) → x smiles.

The facts into FOPL are:

1. ∀x (Graduating(x) → Happy(x))
2. ∀x (Happy(x) → Smile(x))
3. ∃x Graduating(x)

Negating the statement which has to be proved:

4. ¬ (∃x Smile(x))

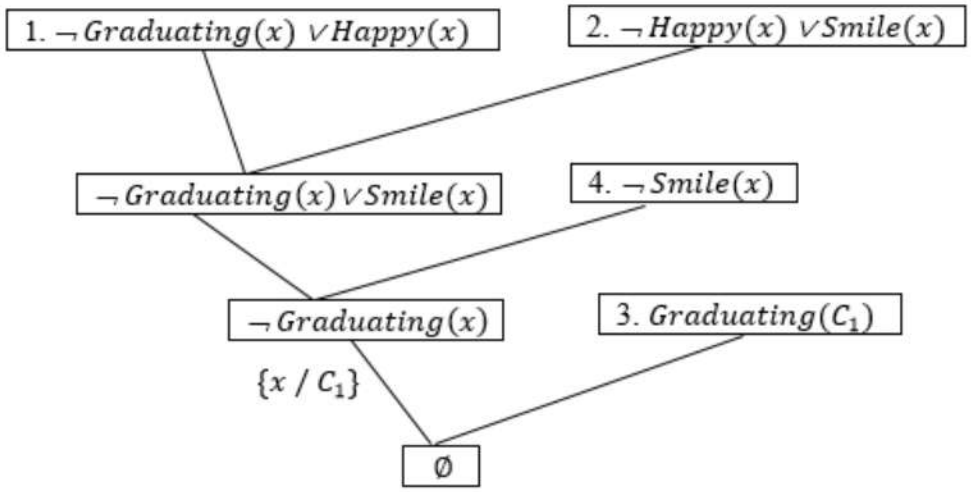
Converting to CNF,

1. ∀x (Graduating(x) → Happy(x))
   ∀x (¬Graduating(x) ∨ Happy(x))
   ¬Graduating(x) ∨ Happy(x)
2. ∀x (Happy(x) →Smile(x))
   ∀x (¬Happy(x) ∨ Smile(x))
   ¬Happy(x) ∨ Smile(x)
3. ∃x Graduating(x)
   Graduating(C_1) Where C_1 is Skolemization Constant
4. ¬ (∃x Smile(x))
   ∀x ¬Smile(x)
   ¬Smile(x)

The KB contains following sentences in CNF,

1. ¬Graduating(x) ∨ Happy(x)
2. ¬Happy(x) ∨ Smile(x)
3. Graduating(C_1)
4. ¬Smile(x)

Now, Using resolution



Since an empty clause has been deduced we say that our assumption is wrong and hence we have proved "Someone is smilng".

Q. All oversmart persons are stupid. Children of stupid presons are naughty. Ram is children of Hari. Hari is oversmart. Show that Ram is naughty. Using FOPL based resolution method.

Solution:

Let's assume, for FOPL, the following predicates:

Oversmart(x) → x is oversmart person.

Stupid(x) → x is stupid person.
Children(x, y) → y is children of x.
Naughty(x) → x is naughty.

The facts are transformed to FOPL as:

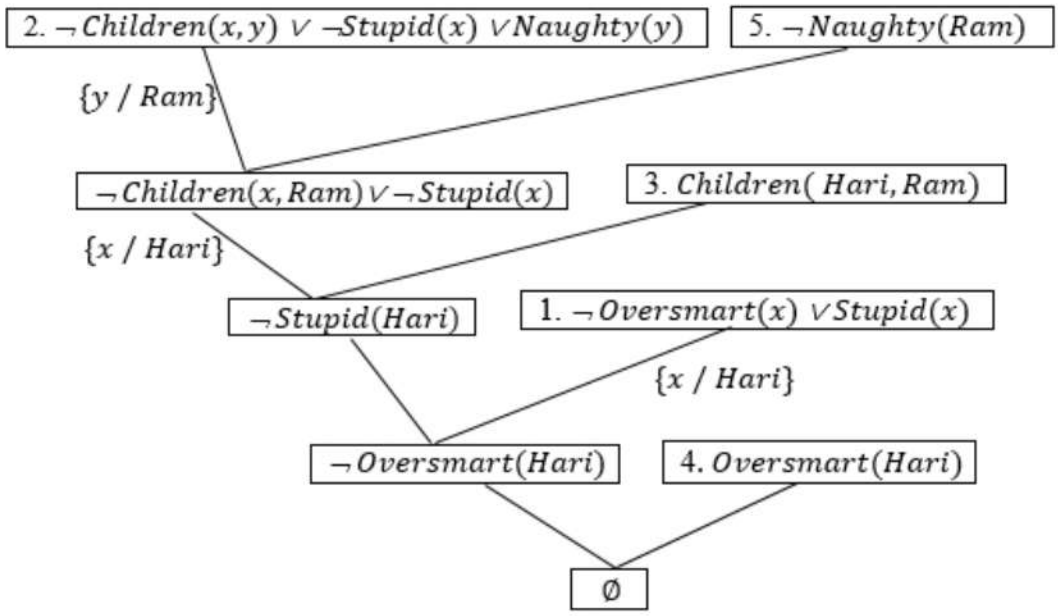
1. ∀x [Oversmart(x) → Stupid(x)]
2. ∀x∀y [(Children(x, y) ∧ Stupid(x)) ⇒ Naughty(y)]
3. Children(Hari, Ram)
4. Oversmart(Hari)
5. ¬Naughty(Ram) ***(Negation of the statement to be proved)***

Converting to CNF,

1. ∀x [Oversmart(x) → Stupid(x)]
   ∀x [¬Oversmart(x) ∨ Stupid(x)]
   ¬ Oversmart(x) ∨ Stupid(x)
2. ∀x∀y [(Children(x, y) ∧ Stupid(x)) → Naughty(y)]
   ∀x∀y [¬ (Children(x, y) ∧ Stupid(x)) ∨ Naughty(y)]
   ¬ Children(x, y) ∨ ¬Stupid(x) ∨ Naughty(y)
3. Children(Hari, Ram)
4. Oversmart(Hari)
5. ¬Naughty(Ram)

The KB contains following sentences in CNF,

1. ¬ Oversmart(x) ∨ Stupid(x)
2. ¬ Children(x, y) ∨ ¬Stupid(x) ∨ Naughty(y)
3. Children(Hari, Ram)
4. Oversmart(Hari)
5. ¬Naughty(Ram)



Since an empty clause has been deduced we say that our assumption is wrong and hence we have proved "Ram is naughty".

***Note**: For additional examples, including FOPL to CNF conversion and proofs by resolution, please refer to the class notes.*

Propositional Logic Vs. First-Order Predicate Logic (FOPL)

Propositional Logic	Predicate Logic
Propositional logic is the logic that deals with a collection of declarative statements which have a truth value, true or false	Predicate logic is an expression consisting of variables with a specified domain. It consists of objects, relations and functions between the objects.
It is the basic and most widely used logic.	It is an extension of propositional logic covering predicates and quantification.
A proposition has a specific truth value, either true or false.	A predicate's truth value depends on the variables' value.
Scope analysis is not done in propositional logic.	Predicate logic helps analyze the scope of the subject over the predicate. There are two quantifiers: Universal Quantifier (∀) depicts for all and Existential Quantifier (∃) depicting there exists some.
Propositions are combined with Logical Operators or Logical Connectives like Negation(¬), Disjunction(∧), Conjunction(∨) ,Implication(⇒),Bi-Conditional or Double Implication(⇔).	Predicate Logic adds by introducing quantifiers to the existing proposition.
It cannot deal with sets of entities.	It can deal with set of entities with the help of quantifiers.

Definite Clause and Horn Clause

- ***Definite Clause**:* A clause which is a disjunction of literals with ***exactly one positive i.e. unnegated literal*** is known as a definite clause or ***strict horn clause***. For example,
  - $\neg p \vee q$ is a definite clause.
  - $p \vee q$ and $\neg p \vee \neg q$ are not definite clauses.

- ***Horn Clause**:* A clause which is a disjunction of literals with ***at most one positive literal*** is known as horn clause. Hence all the definite clauses are horn clause. For example,
  - $\neg p \vee q$ and $\neg p \vee \neg q$ are horn clauses.
  - $p \vee q$ is not a horn clause.