CO-2

Syllabus: CO-2:

Constraint Satisfaction: Problem formulation, Constraint propagation, Backtracking algorithms Knowledge Engineering: propositional Logic, Predicate Logic, Inferencing through propositional and Predicate Logic: Introduction, Inferencing rules, Inferencing Mechanisms: Entitlement, Resolution, Lifting, Reasoning, Implementing inferencing: Forward Checking and Backward Chaining, Introduction to probability theory, Introduction to uncertainty Bayes Theorem

Constraint Satisfaction problem(CSP)

Finding a solution that meets a set of constraints is the goal of constraint satisfaction problems (CSPs), a type of AI issue. Finding values for a group of variables that fulfill a set of restrictions or rules is the aim of constraint satisfaction problems.

There are mainly three basic components in the constraint satisfaction problem:

Variables: The things that need to be determined are variables. Variables in a CSP are the objects that must have values assigned to them in order to satisfy a particular set of constraints. Boolean, integer, and categorical variables are just a few examples of the various types of variables, for instance, could stand in for the many puzzle cells that need to be filled with numbers in a sudoku puzzle.

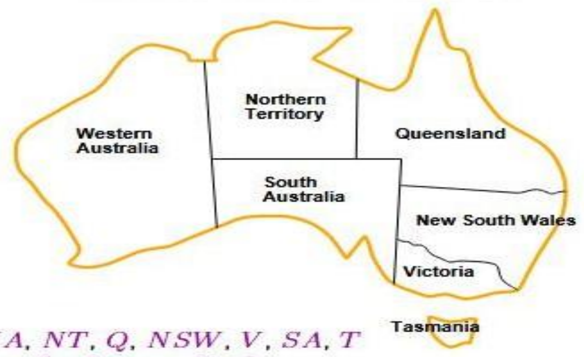
Domains: The range of potential values that a variable can have is represented by domains. Depending on the issue, a domain may be finite or limitless. For instance, in Sudoku, the set of numbers from 1 to 9 can serve as the domain of a variable representing a problem cell.

Constraints: The guidelines that control how variables relate to one another are known as constraints. Constraints in a CSP define the ranges of possible values for variables. Unary constraints, binary constraints, and higher-order constraints are only a few examples of the various sorts of constraints. For instance, in a sudoku problem, the restrictions might be that each row, column, and 3×3 box can only have one instance of each number from 1 to 9.

Constraint Satisfaction Problems (CSP) representation:

- The finite set of variables V_1, V_2, V_3……………………V_n.
- Non-empty domain for every single variable D_1, D_2, D_3…………………D_n.
- The finite set of constraints C_1, C_2……………, Cm.

Example: Map-Coloring



Variables WA, NT, Q, NSW, V, SA, T
Domains $D_i = \{red, green, blue\}$
Constraints: adjacent regions must have different colors
e.g., $WA \neq NT$ (if the language allows this), or
$(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), \ldots\}$

Example: Map-Coloring contd.

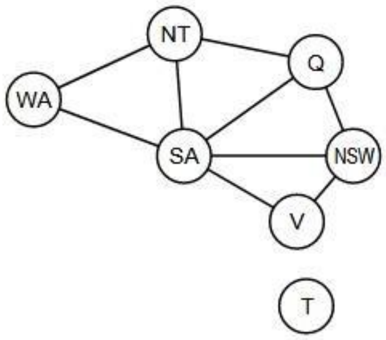
Western Australia
Northern Territory
Queensland
South Australia
New South Wales
Victoria
Tasmania

Solutions are assignments satisfying all constraints, e.g.,
$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$

Constraint graph

Binary CSP: each constraint relates at most two variables

Constraint graph: nodes are variables, arcs show constraints



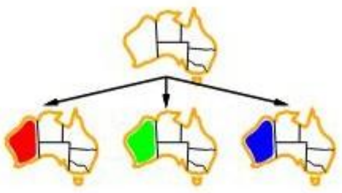
General-purpose CSP algorithms use the graph structure
to speed up search. E.g., Tasmania is an independent subproblem!

Backtracking search

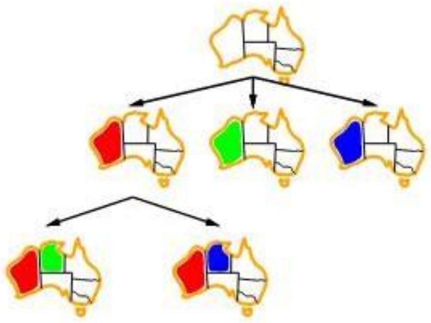
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
   return RECURSIVE-BACKTRACKING({ }, csp)
function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment given CONSTRAINTS[csp] then
             add {var = value} to assignment
             result ← RECURSIVE-BACKTRACKING(assignment, csp)
             if result ≠ failure then return result
             remove {var = value} from assignment
   return failure
```

Backtracking example

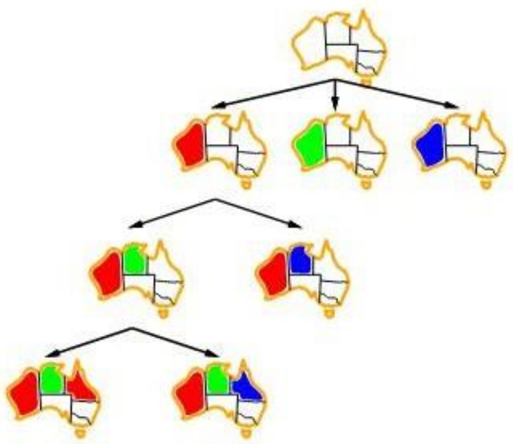
Backtracking example



Backtracking example

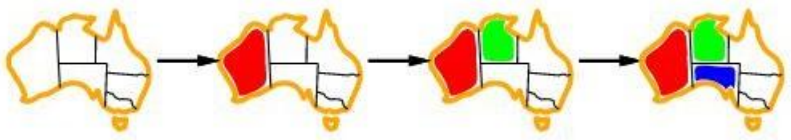


Backtracking example



Minimum remaining values

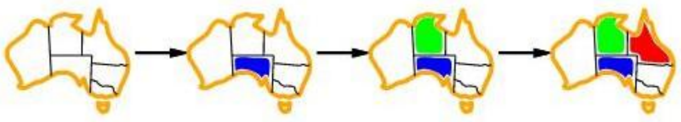
Minimum remaining values (MRV):
  choose the variable with the fewest legal values



Degree heuristic

Tie-breaker among MRV variables

Degree heuristic:
  choose the variable with the most constraints on remaining variables



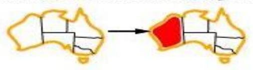
Forward checking

Idea: Keep track of remaining legal values for unassigned variables
  Terminate search when any variable has no legal values

WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦

Forward checking

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values



WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦

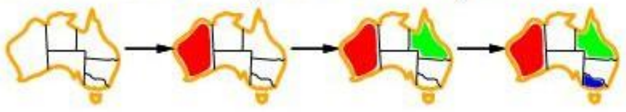
Forward checking

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦

Forward checking

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values



WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦
🟥	🟦	🟩	🟥	🟦		🟥🟩🟦

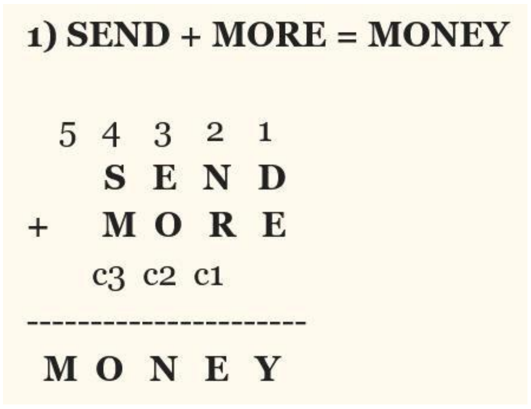
Crypt Arithmetic problem:

The **Crypt-Arithmetic problem in Artificial Intelligence** is a type of encryption problem in which the written message in an alphabetical form which is easily readable and understandable is converted into a numeric form which is neither easily readable nor understandable. In simpler words, the crypt-arithmetic problem deals with the converting of the message from the readable plain text to the non-readable ciphertext. The constraints which this problem follows during the conversion is as follows:

1. A number 0-9 is assigned to a particular alphabet.
2. Each different alphabet has a unique number.
3. All the same, alphabets have the same numbers.
4. The numbers should satisfy all the operations that any normal number does.

Let us take an example of the message:

SEND+MORE=MONEY

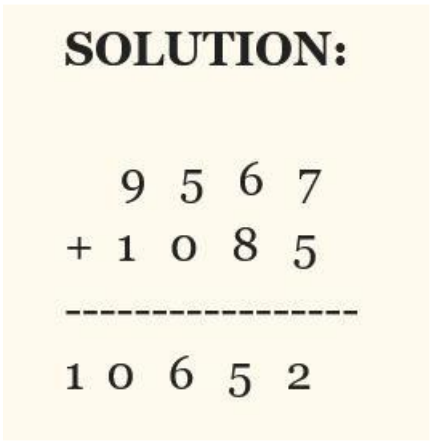


1) SEND + MORE = MONEY

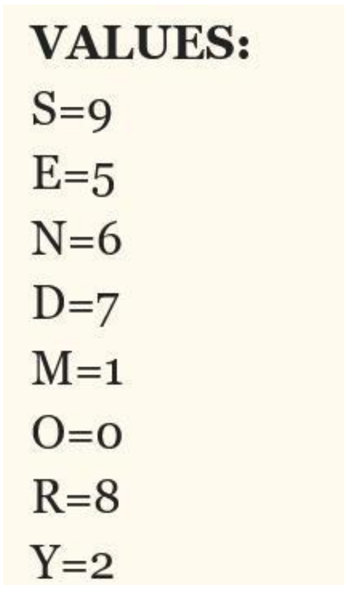
```
  5 4  3  2  1
    S  E  N  D
+   M  O  R  E
   c3 c2 c1
----------------------
  M  O  N  E  Y
```

1. From Column 5, **M=1**, since it is only carry-over possible from sum of 2 single digit number in column 4.
2. To produce a carry from column 4 to column 5 'S + M' is atleast 9 so **'S=8or9'** so **'S+M=9or10'** & so **'O = 0 or 1'**. But 'M=1', so **'O = 0'**.
3. If there is carry from Column 3 to 4 then 'E=9' & so 'N=0'. But 'O = 0' so there is no carry & **'S=9'** & **'c3=0'**.
4. If there is no carry from column 2 to 3 then 'E=N' which is impossible, therefore there is carry & **'N=E+1'** & **'c2=1'**.
5. If there is carry from column 1 to 2 then **'N+R=E mod 10'** & 'N=E+1' so 'E+1+R=E mod 10', so 'R=9' but 'S=9', so there must be carry from column 1 to 2. Therefore **'c1=1'** & **'R=8'**.
6. To produce carry 'c1=1' from column 1 to 2, we must have 'D+E=10+Y' as Y cannot be 0/1 so D+E is atleast 12. As D is atmost 7 & E is atleast 5 (D cannot be 8 or 9 as it is already assigned). N is atmost 7 & 'N=E+1' so 'E=5or6'.
7. If E were 6 & D+E atleast 12 then D would be 7, but 'N=E+1' & N would also be 7 which is impossible. Therefore **'E=5'** & **'N=6'**.

7. D+E is atleast 12 for that we get **'D=7'** & **'Y=2'**.


SOLUTION:

```
   9 5 6 7
 + 1 0 8 5
----------------
 1 0 6 5 2
```



VALUES:

S=9

E=5

N=6

D=7

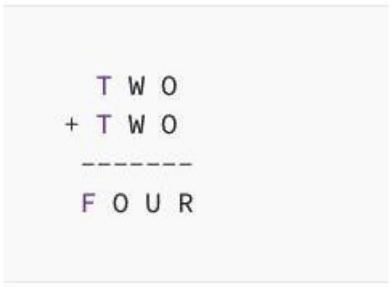
M=1

O=0

R=8

Y=2

Solve the following crypt Arithmetic Problem



```
  T W O
+ T W O
 -------
 F O U R
```

Constraints: To solve this cryptarithmetic problem, we must adhere to several constraints:

1. Distinct Digits: Each letter (T, W, O, F, U, and R) must represent a distinct digit from 0 to 9. In other words, no two letters can have the same assigned digit.
2. No Leading Zeros: Since no number can have leading zeroes, the letter 'T' cannot be assigned the digit 0, and 'F' cannot be assigned the digit 0 either.
3. Correct Arithmetic Operation: The equation must hold true, which means that the sum of 'TWO' and 'TWO' should equal 'FOUR' when digits are substituted for letters.

Determine the digit represented by each letter to make the addition correct. F =

N =

O =

R =

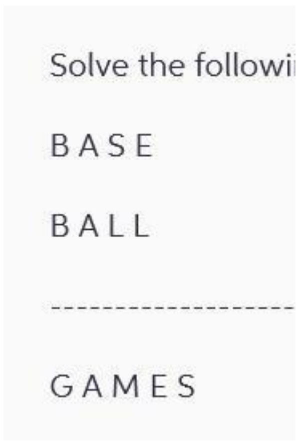
T =

U =

W =

```
    7  6  5
 +  7  6  5
 ------------
  1 5  3  0
```

Solve the following crypt Arithmetic Problem



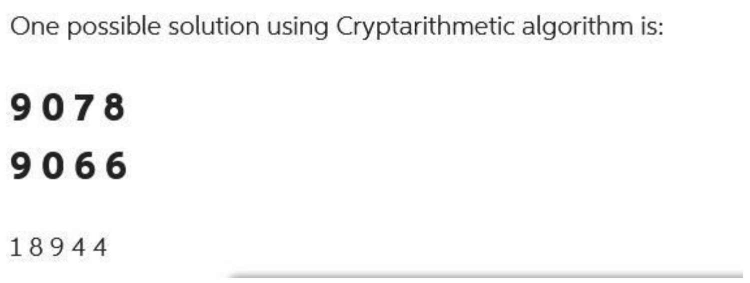
Solve the followi

```
B A S E

B A L L

-------------------

G A M E S
```

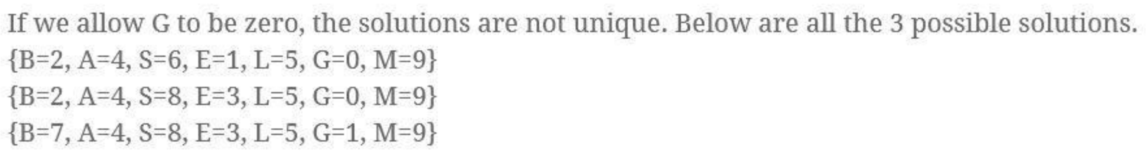


One possible solution using Cryptarithmetic algorithm is:

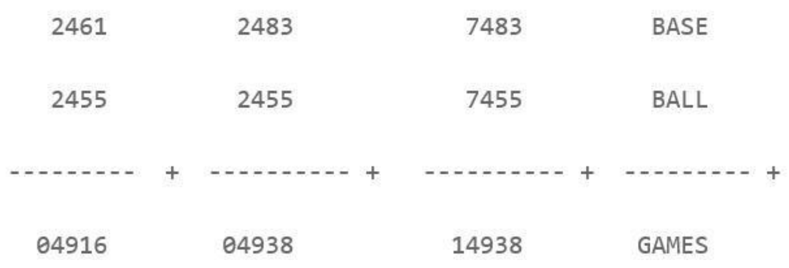
9 0 7 8

9 0 6 6

1 8 9 4 4



If we allow G to be zero, the solutions are not unique. Below are all the 3 possible solutions.
{B=2, A=4, S=6, E=1, L=5, G=0, M=9}
{B=2, A=4, S=8, E=3, L=5, G=0, M=9}
{B=7, A=4, S=8, E=3, L=5, G=1, M=9}



```
   2461         2483           7483        BASE

   2455         2455           7455        BALL

---------  +  ---------- +   ---------- +  --------- +

  04916        04938          14938       GAMES
```

Knowledge and Reasoning

What is knowledge representation?

Humans are best at understanding, reasoning, and interpreting knowledge. Human knows things, which is knowledge and as per their knowledge they perform various actions in the real world. **But how machines do all these things comes under knowledge representation and reasoning**. Hence we can describe Knowledge representation as following:

- Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which concerned with AI agents thinking and how thinking contributes to intelligent behavior of agents.
- It is responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve the complex real world problems such as diagnosis a medical condition or communicating with humans in natural language.
- It is also a way which describes how we can represent knowledge in artificial intelligence. Knowledge representation is not just storing data into some database, but it also enables an intelligent machine to learn from that knowledge and experiences so that it can behave intelligently like a human.

What to Represent:

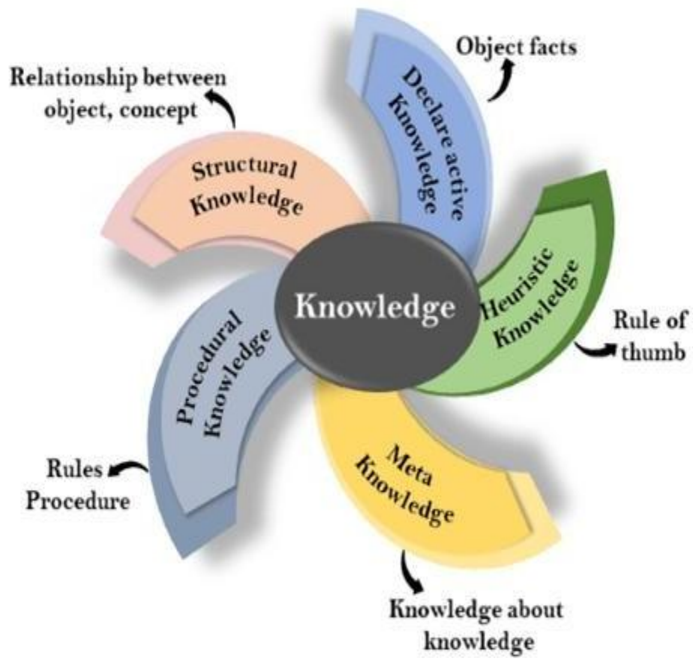
Following are the kind of knowledge which needs to be represented in AI systems:

- **Object:** All the facts about objects in our world domain. E.g., Guitars contains strings, trumpets are brass instruments.
- **Events:** Events are the actions which occur in our world.
- **Performance:** It describe behavior which involves knowledge about how to do things.
- **Meta-knowledge:** It is knowledge about what we know.
- **Facts:** Facts are the truths about the real world and what we represent.
- **Knowledge-Base:** The central component of the knowledge-based agents is the knowledge base. It is represented as KB. The Knowledgebase is a group of the Sentences (Here, sentences are used as a technical term and not identical with the English language).

Knowledge: Knowledge is awareness or familiarity gained by experiences of facts, data, and situations. Following are the types of knowledge in artificial intelligence:

Types of knowledge

Following are the various types of knowledge:



1. Declarative Knowledge:

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

2. Procedural Knowledge

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

3. Meta-knowledge:

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

4. Heuristic knowledge:

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

5. Structural knowledge:

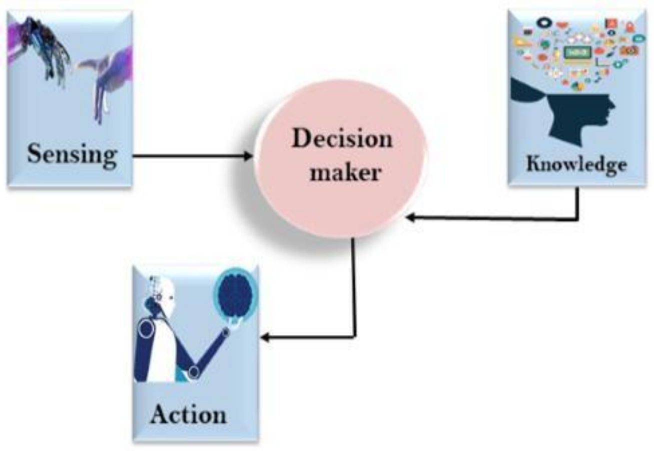
- Structural knowledge is basic knowledge to problem-solving.
- It describes relationships between various concepts such as kind of, part of, and grouping of something.
- It describes the relationship that exists between concepts or objects.

The relation between knowledge and intelligence:

Knowledge of real-worlds plays a vital role in intelligence and same for creating artificial intelligence. Knowledge plays an important role in demonstrating intelligent behavior in AI agents. An agent is only able to accurately act on some input when he has some knowledge or experience about that input.

Let's suppose if you met some person who is speaking in a language which you don't know, then how you will able to act on that. The same thing applies to the intelligent behavior of the agents.

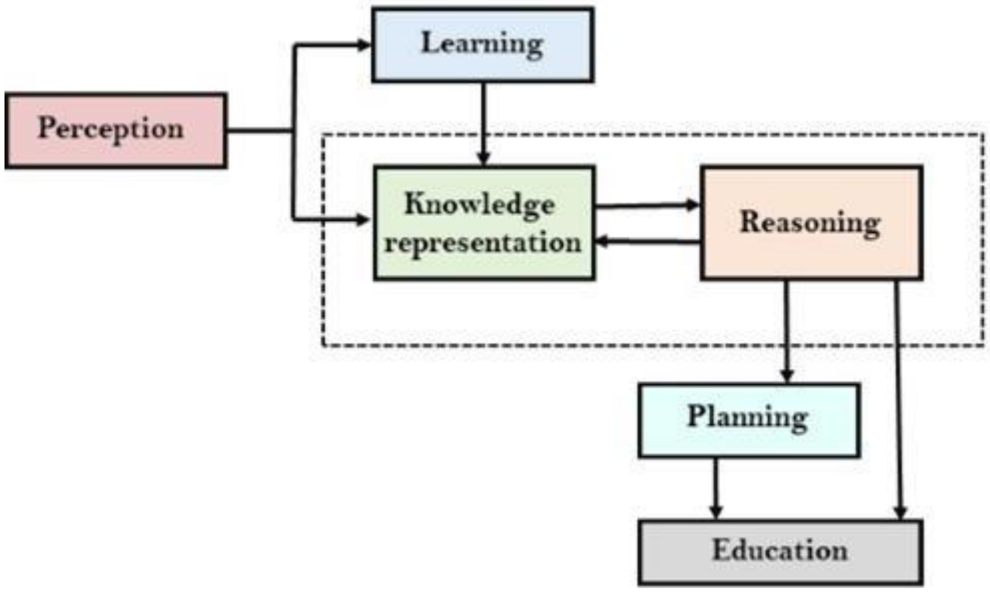
As we can see in below diagram, there is one decision maker which act by sensing the environment and using knowledge. But if the knowledge part will not present then, it cannot display intelligent behavior.



AI knowledge **cycle:**

An Artificial intelligence system has the following components for displaying intelligent behavior:

- Perception
- Learning
- Knowledge Representation and Reasoning
- Planning
- Execution



the above diagram is showing how an AI system can interact with the real world and what components help it to show intelligence. AI system has Perception component by which it retrieves information from its environment. It can be visual, audio or another form of sensory input. The learning component is responsible for learning from data captured by Perception comportment. In the complete cycle, the main components are knowledge representation and Reasoning. These two components are involved in showing the intelligence in machine-like humans. These two components are independent with each other but also coupled together. The planning and execution depend on analysis of Knowledge representation and reasoning. Approaches to knowledge representation:

There are mainly four approaches to knowledge representation, which are given below:

1. Simple relational knowledge:

- It is the simplest way of storing facts which uses the relational method, and each fact about a set of the object is set out systematically in columns.
- This approach of knowledge representation is famous in database systems where the relationship between different entities is represented.
- This approach has little opportunity for inference.

Knowledge and Reasoning in Artificial Intelligence

A Knowledge Based Agent in Artificial Intelligence has two levels: Knowledge Base (KB) and Inference Engine.

1. Knowledge Base- It is the base level of an agent, which consist of domain specific content. In this level agent has facts or information about the surrounding environment in which they are working. It does not consider the actual implementation.

2. Implementation level- It consists of domain independent algorithms. At this level, agents can recognize the data structures used in the knowledge base and algorithms which use them. For example, propositional logic and resolution. Knowledge based agents are crucial to use in partially observable environments. Before choosing any action, knowledge based agents make use of the existing knowledge along with the current inputs from the environment in order to infer hidden aspects of the current state.

An agent makes use of TELL and ASK mechanism.

-TELL the agent, about surrounding environment what it needs to know in order to perform some action. TELL mechanism is similar to taking input for a system.

-Then the agent ASKs itself what action should be carried out to get desired output. ASK mechanism is similar to producing output for a system. However, ASK mechanism makes use of the knowledge base to decide what it should do.

TELL(K): Is a function that adds knowledge K to the knowledge base.

ASK(K): Is a function that queries the agent about the truth of K.

An agent carries out the following operations: First, it TELLs the knowledge base about facts/ information it perceives with the help of sensors. Then, it ASKs the knowledge base what action should be carried out based on the input it has received. Then it performs the selected action with the help of effectors.

Representation of knowledge in AI

The objective here is to express knowledge in a computer traceable form such that it can be used to help agents perform well. To make programs capable of representing and using this knowledge in a more explicit way. There is a set of facts which the program will use to figure out how to solve the problem. One approaches to deal with knowledge is based on Logic.

P**ropositional logic**

It is a simple knowledge representation language. It works at the sentential level. The sentences here are called propositions. We do not go within the individual sentences and discuss their meaning. Propositional logic is unambiguous and is also called Boolean logic as the sentences or propositions return a true or false value. Further, all other logic like First Order Logic are built on top of Propositional Logic. For instance,

Grass is green.

It is Sunday.

2 + 2=4.

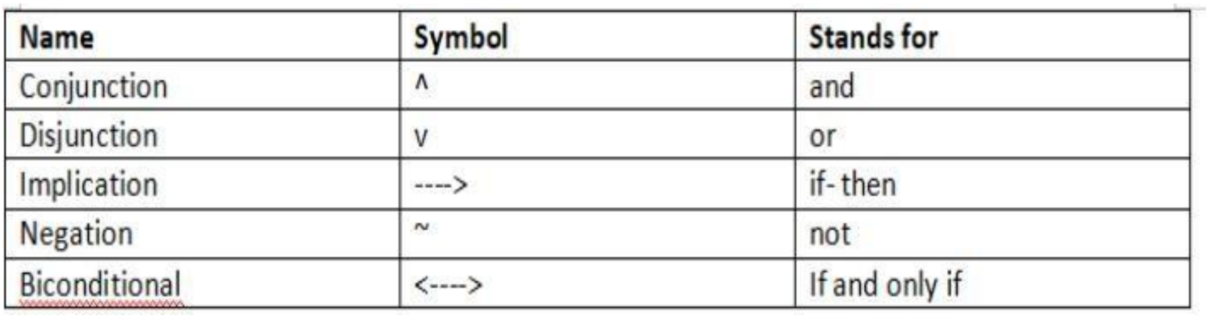
These are all valid propositions, whereas

Close the door.

Is it Monday today?

x=x are invalid propositions

Atomic propositions are simple propositions and Compound propositions are constructed by combining atomic propositions using logical connectives and parenthesis. For instance, It is Sunday and it is a holiday. Logical connectives are used in propositional logic to connect two sentences logically:



Name	Symbol	Stands for
Conjunction	^	and
Disjunction	v	or
Implication	--->	if- then
Negation	~	not
Biconditional	<--->	If and only if

Let’s look at this with the help of an example. If it is hot and humid, then it is raining.

Here P= It is hot,

Q= It is humid,

R=It is raining

This can be represented as:

$(P \wedge Q) \rightarrow R$

Properties of Propositional Logic

1. Commutativity

2. Associativity

3. Distributive property

4. DeMorgan’s law

5. Double negation elimination

6. Identity element

However, Propositional Logic has limited expressive power. It cannot be used to represent specializations, generalizations, or patterns. Elements like all, some and none cannot be expressed using propositional logic. For example, All mushrooms are brown, some grapes are sweet, and more.

Predicate logic or First Order Logic

It is another knowledge representation language built on top of propositional logic. It is more expressive as compared to propositional logic and supports temporal information. It is also called higher order logic and can support complex statements.

Components

- User defines primitives:

1. Constant symbols- terms with fixed value which belong to the domain. Example: Tara, 2.

2. Function symbols- mapping individuals to individuals. Example: Mary is the mother of Tara. mother_of(Tara)=Mary

3. Predicate symbols- mapping individuals to truth values. Example: 4 is greater than 2. greater(4,2)

- FOL supports these primitives:

1. Variable symbols- x,y

2. Connectives- conjunction, disjunction, implication, negation, biconditional

3. Quantifiers

A. Universal — It stands for "for all" and is represented by the symbol ∀. It corresponds to conjunction 'and.' Example: All dolphins are mammals.

∀x: dolphin(x) — -> mammal(x)

B. Existential- It stands for "there exists". It is represented by the symbol ∃ and corresponds to disjunction 'or.' Example: Some mammals lay eggs.

∃x: mammal(x) — -> lay_eggs(x)

C. Equal: X=Y

In predicate logic, a rule has two parts: predecessor and successor. If the predecessor is true, the successor is true. It uses the implication symbol. It represents if-then type of sentences. Example: If the bag is of pink color, I will buy it. colour(bag, pink) — — -> buy(bag)

First-Order Logic in Artificial intelligence

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

- **"Some humans are intelligent", or**
- **"Sachin likes cricket."**

First-Order logic:

- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in a concise way.
- First-order logic is also known as **Predicate logic or First-order predicate logic**. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.
- First-order logic (like natural language) does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:
    - **Objects:** A, B, people, numbers, colors, wars, theories, squares, pits, wumpus, ...
    - **Relations: It can be unary relation such as:** red, round, is adjacent, **or n-any relation such as:** the sister of, brother of, has color, comes between
    - **Function:** Father of, best friend, third inning of, end of, ...
- As a natural language, first-order logic also has two main parts:
    - **Syntax**
    - **Semantics**

Syntax of First-Order logic:

The syntax of FOL determines which collection of symbols is a logical expression in first-order logic. The basic syntactic elements of first-order logic are symbols. We write statements in short-hand notation in FOL.

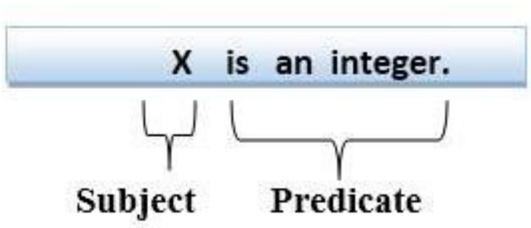
Basic Elements of First-order logic:

Following are the basic elements of FOL syntax:

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf, ...
Connectives	$\land$, $\lor$, $\neg$, $\Rightarrow$, $\Leftrightarrow$
Equality	==

Quantifier ∀, ∃

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



Quantifiers in First-order logic:

- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
- These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifier:
    1. **Universal Quantifier, (for all, everyone, everything)**
    2. **Existential quantifier, (for some, at least one).**

Universal Quantifier:

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

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

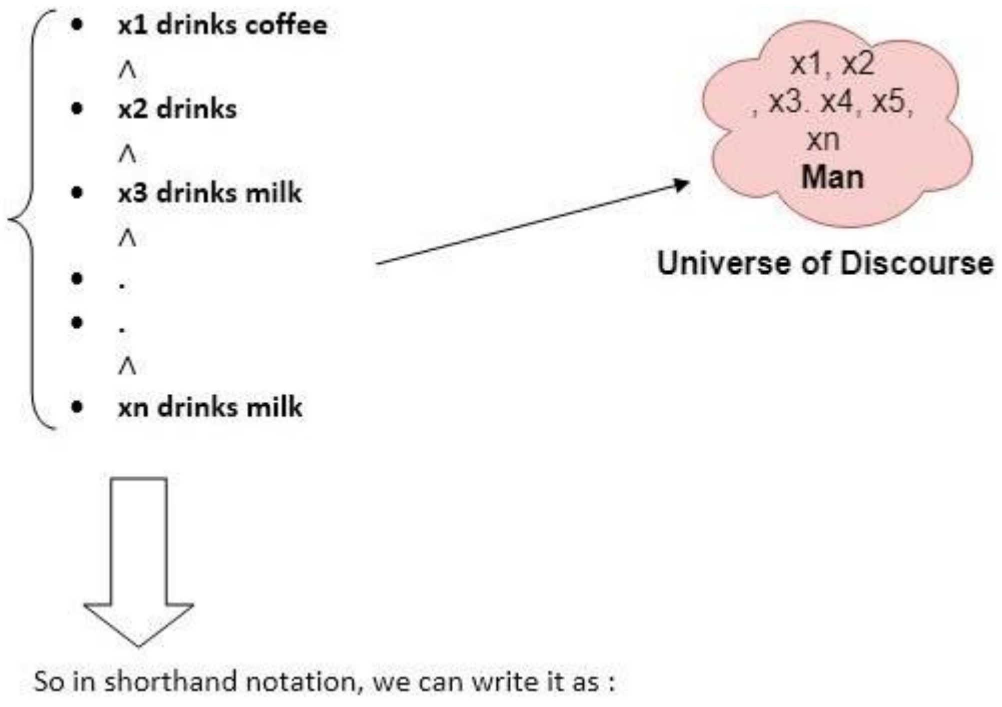
Note: In universal quantifier we use implication "→".

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

- **For all x**
- **For each x**
- **For every x.**

Example:

All man drink coffee.



∀x man(x) → drink (x, coffee).

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

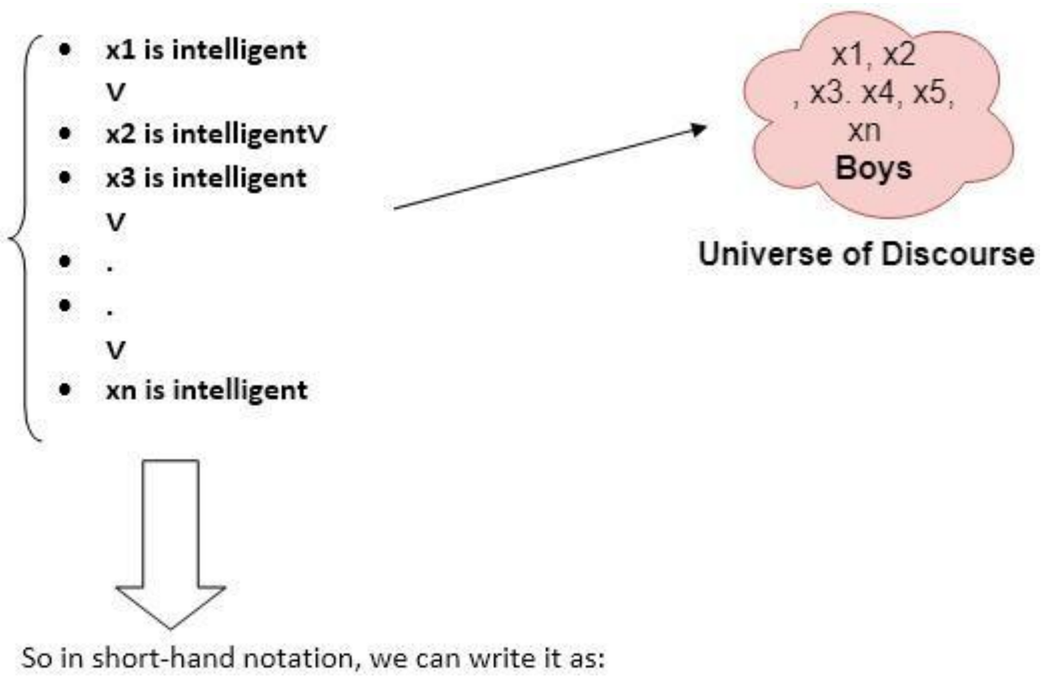
Existential Quantifier:

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

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

Note: In Existential quantifier we always use AND or Conjunction symbol (∧).

Example:
Some boys are intelligent.



Some Examples of FOL using quantifier:

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

$$\forall x \ \text{bird}(x) \rightarrow \text{fly}(x).$$

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

$$\forall x \ \text{man}(x) \rightarrow \text{respects}(x, \text{parent}).$$

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

$$\exists x \ \text{boys}(x) \rightarrow \text{play}(x, \text{cricket}).$$

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

$$\neg\forall (x) \ [\ \text{student}(x) \rightarrow \text{like}(x, \text{Mathematics}) \land \text{like}(x, \text{Science})].$$

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

$$\exists (x) \ [\ \text{student}(x) \rightarrow \text{failed}(x, \text{Mathematics}) \land \forall (y) \ [\neg(x{=}{=}y) \land \text{student}(y) \rightarrow \neg\text{failed}(x, \text{Mathematics})].$$

What is 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.
- Let Ψ_1 and Ψ_2 be two atomic sentences and be a unifier such that, $\mathbf{\Psi_1 = \Psi_2}$, then it can be expressed as **UNIFY(Ψ_1, Ψ_2)**.
- **Example: Find the MGU for Unify{King(x), King(John)}**

Let Ψ_1 = King(x), Ψ_2 = King(John),

Substitution θ = {John/x} is a unifier for these atoms and applying this substitution, and both expressions will be identical.

- The UNIFY algorithm is used for unification, which takes two atomic sentences and returns a unifier for those sentences (If any exist).
- Unification is a key component of all first-order inference algorithms.
- It returns fail if the expressions do not match with each other.
- The substitution variables are called Most General Unifier or MGU.

E.g. Let's say there are two different expressions, **P(x, y), and P(a, f(z))**.

In this example, we need to make both above statements identical to each other. For this, we will perform the substitution.

P(x, y)......... (i)
P(a, f(z)) .... (ii)

- Substitute x with a, and y with f(z) in the first expression, and it will be represented as **a/x** and f(z)/y.
- With both the substitutions, the first expression will be identical to the second expression and the substitution set will be: **[a/x, f(z)/y]**.

Conditions for Unification:

Following are some basic conditions for unification:

- Predicate symbol must be same, atoms or expression with different predicate symbol can never be unified.
- Number of Arguments in both expressions must be identical.
- Unification will fail if there are two similar variables present in the same expression.

Unification Algorithm:

Algorithm: Unify(Ψ_1, Ψ_2)

```
Step. 1: If $\Psi_1$ or $\Psi_2$ is a variable or constant, then:
        a) If $\Psi_1$ or $\Psi_2$ are identical, then return NIL.
        b) Else if $\Psi_1$is a variable,
                a. then if $\Psi_1$ occurs in $\Psi_2$, then return FAILURE
                b. Else return { ($\Psi_2$/ $\Psi_1$)}.
        c) Else if $\Psi_2$ is a variable,
                a. If $\Psi_2$ occurs in $\Psi_1$ then return FAILURE,
                b. Else return {( $\Psi_1$/ $\Psi_2$)}.
        d) Else return FAILURE.
Step.2: If the initial Predicate symbol in $\Psi_1$ and $\Psi_2$ are not same, then return FAILURE.
Step. 3: IF $\Psi_1$ and $\Psi_2$ have a different number of arguments, then return FAILURE.
Step. 4: Set Substitution set(SUBST) to NIL.
Step. 5: For i=1 to the number of elements in $\Psi_1$.
        a) Call Unify function with the ith element of $\Psi_1$ and ith element of $\Psi_2$, and put the result into S.
        b) If S = failure then returns Failure
        c) If S $\neq$ NIL then do,
                a. Apply S to the remainder of both L1 and L2.
                b. SUBST= APPEND(S, SUBST).
Step.6: Return SUBST.
```

1. Find the MGU of {p(f(a), g(Y)) and p(X, X)}

Sol: S_0 => Here, Ψ_1 = p(f(a), g(Y)), and Ψ_2 = p(X, X)
SUBST θ= {f(a) / X}
S1 => Ψ_1 = p(f(a), g(Y)), and Ψ_2 = p(f(a), f(a))
SUBST θ= {f(a) / g(y)}, **Unification failed**.

Unification is not possible for these expressions.

2. Find the MGU of {p(b, X, f(g(Z))) and p(Z, f(Y), f(Y))}

Here, Ψ_1 = p(b, X, f(g(Z))) , and Ψ_2 = p(Z, f(Y), f(Y))
S_0 => { p(b, X, f(g(Z))); p(Z, f(Y), f(Y))}
SUBST θ={b/Z}

S_1 => { p(b, X, f(g(b))); p(b, f(Y), f(Y))}
SUBST θ={f(Y) /X}

S_2 => { p(b, f(Y), f(g(b))); p(b, f(Y), f(Y))}
SUBST θ= {g(b) /Y}

S_2 => { p(b, f(g(b)), f(g(b)); p(b, f(g(b)), f(g(b))} **Unified Successfully.**
And Unifier = { b/Z, f(Y) /X , g(b) /Y}.

3. Find the MGU of {p (X, X), and p (Z, f(Z))}

Here, Ψ_1 = {p (X, X), and Ψ_2 = p (Z, f(Z))
S_0 => {p (X, X), p (Z, f(Z))}
SUBST θ= {X/Z}
S1 => {p (Z, Z), p (Z, f(Z))}
SUBST θ= {f(Z) / Z}, **Unification Failed**.

Hence, unification is not possible for these expressions.

4. Find the MGU of UNIFY(prime (11), prime(y))

Here, Ψ_1 = {prime(11) , and Ψ_2 = prime(y)}
S_0 => {prime(11) , prime(y)}
SUBST θ= {11/y}

S_1 => {prime(11) , prime(11)} , **Successfully unified.**
Unifier: {11/y}.

5. Find the MGU of Q(a, g(x, a), f(y)), Q(a, g(f(b), a), x)}

Here, Ψ_1 = Q(a, g(x, a), f(y)), and Ψ_2 = Q(a, g(f(b), a), x)
S_0 => {Q(a, g(x, a), f(y)); Q(a, g(f(b), a), x)}
SUBST θ= {f(b)/x}
S_1 => {Q(a, g(f(b), a), f(y)); Q(a, g(f(b), a), f(b))}

SUBST θ= {b/y}
S_1 => {Q(a, g(f(b), a), f(b)); Q(a, g(f(b), a), f(b))}, **Successfully Unified.**

Unifier: [a/a, f(b)/x, b/y].

Forward Chaining and backward chaining in AI

In artificial intelligence, forward and backward chaining is one of the important topics, but before understanding forward and backward chaining lets first understand that from where these two terms came.

Inference engine:

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

1. **Forward chaining**
2. **Backward chaining**

Horn Clause and Definite clause:

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

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

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

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

It is equivalent to p ∧ q → k.

A. Forward Chaining

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

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

Properties of Forward-Chaining:

- It is a down-up approach, as it moves from bottom to top.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
- Forward-chaining approach is also called as data-driven as we reach to the goal using available data.
- Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

Example:

"As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen."

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

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

Facts Conversion into FOL:

- It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)
  American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)
- Country A has some missiles. **?p Owns(A, p) ∧ Missile(p)**. It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.
  Owns(A, T1).................. **(2)**
  Missile(T1)....................**(3)**
- All of the missiles were sold to country A by Robert.
  ?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A)............. **(4)**
- Missiles are weapons.
  Missile(p) → Weapons (p).................... **(5)**
- Enemy of America is known as hostile.
  Enemy(p, America) →Hostile(p).....................**(6)**
- Country A is an enemy of America.
  Enemy (A, America)......................**(7)**
- Robert is American
  American(Robert)........................ **(8)**

Forward chaining proof:

Step-1:

In the first step we will start with the known facts and will choose the sentences which do not have implications, such as: **American(Robert), Enemy(A, America), Owns(A, T1), and Missile(T1)**. All these facts will be represented as below.

Step-2:

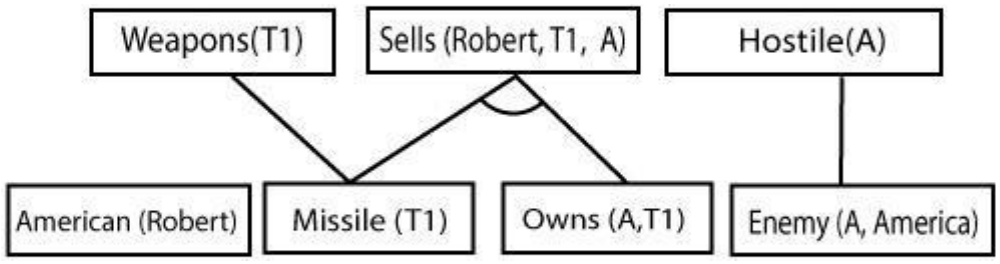
At the second step, we will see those facts which infer from available facts and with satisfied premises.

Rule-(1) does not satisfy premises, so it will not be added in the first iteration.

Rule-(2) and (3) are already added.

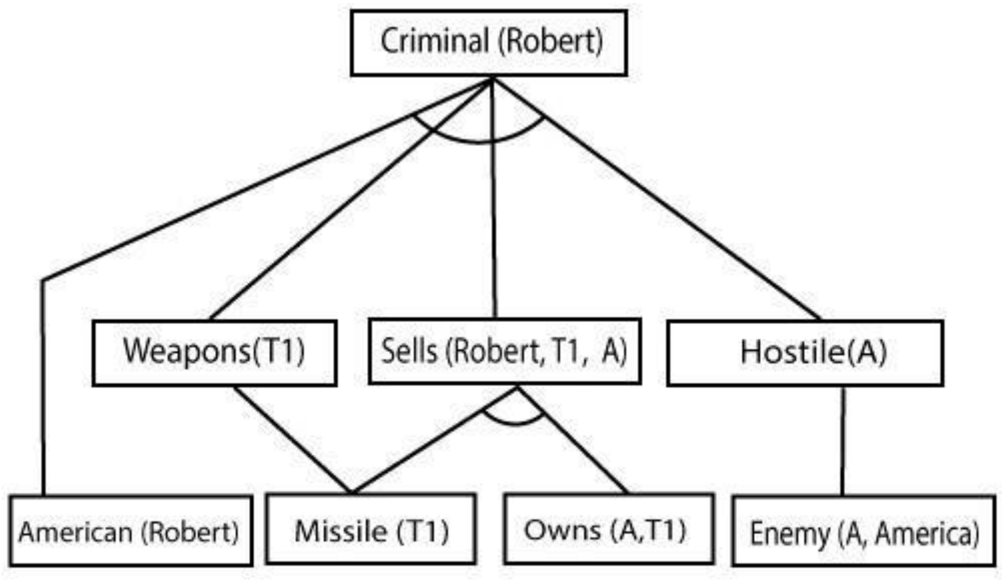
Rule-(4) satisfy with the substitution {p/T1}, **so Sells (Robert, T1, A)** is added, which infers from the conjunction of Rule (2) and (3).

Rule-(6) is satisfied with the substitution(p/A), so Hostile(A) is added and which infers from Rule-(7).



Step-3:

At step-3, as we can check Rule-(1) is satisfied with the substitution **{p/Robert, q/T1, r/A}, so we can add Criminal(Robert)** which infers all the available facts. And hence we reached our goal statement.



Hence it is proved that Robert is Criminal using forward chaining approach.

B. Backward Chaining:

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

Properties of backward chaining:

- It is known as a top-down approach.
- Backward-chaining is based on modus ponens inference rule.
- In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.
- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.
- Backward -chaining algorithm is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.
- The backward-chaining method mostly used a **depth-first search** strategy for proof.

Example:

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

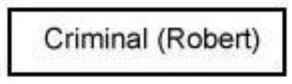
- **American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)**
  Owns(A, T1).........................**(2)**
- **Missile(T1)**
- **?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A)** ..... **(4)**
- **Missile(p) → Weapons (p)**..........................**(5)**
- **Enemy(p, America) →Hostile(p)** ....**(6)**
- **Enemy (A, America)** ....**(7)**
- **American(Robert).**.............................**(8)**

Backward-Chaining proof:

In Backward chaining, we will start with our goal predicate, which is **Criminal(Robert)**, and then infer further rules.

Step-1:

At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is "Robert is Criminal," so following is the predicate of it.

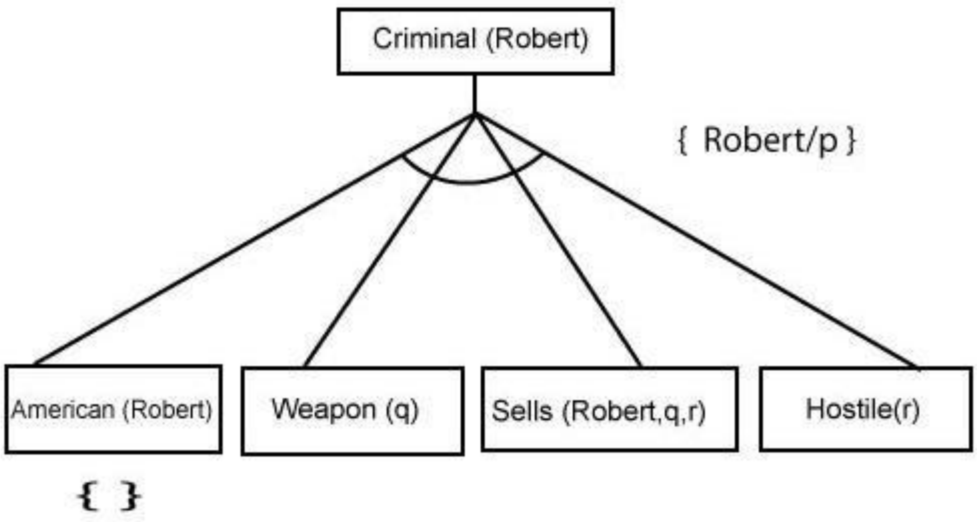


Step-2:

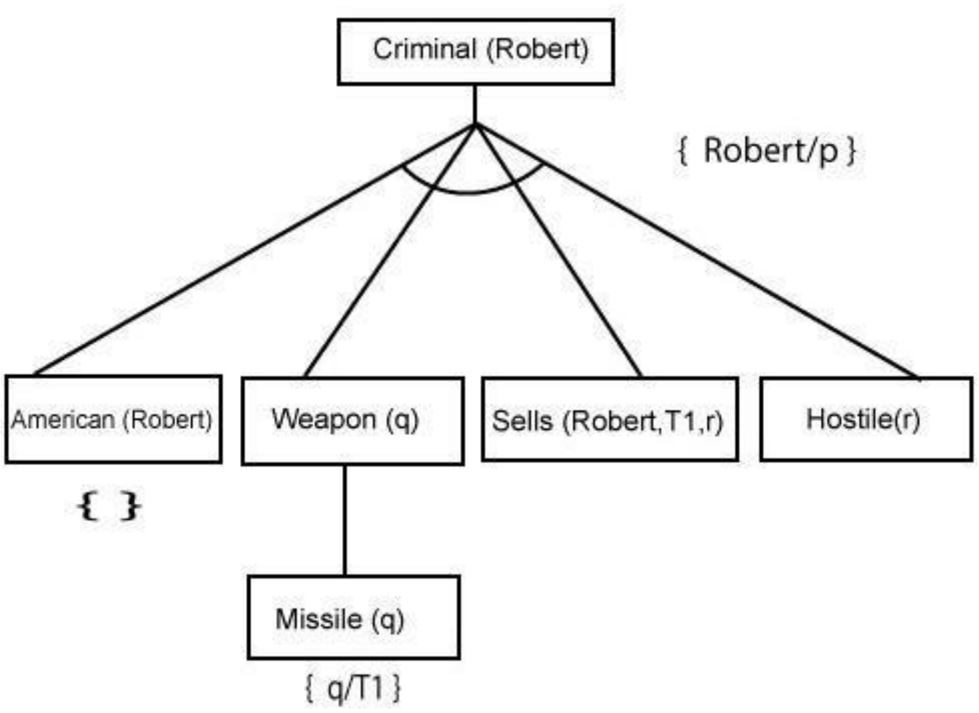
At the second step, we will infer other facts form goal fact which satisfies the rules. So as we can see in Rule-1, the goal predicate Criminal (Robert) is present with substitution

{Robert/P}. So we will add all the conjunctive facts below the first level and will replace p with Robert.

Here we can see American (Robert) is a fact, so it is proved here.

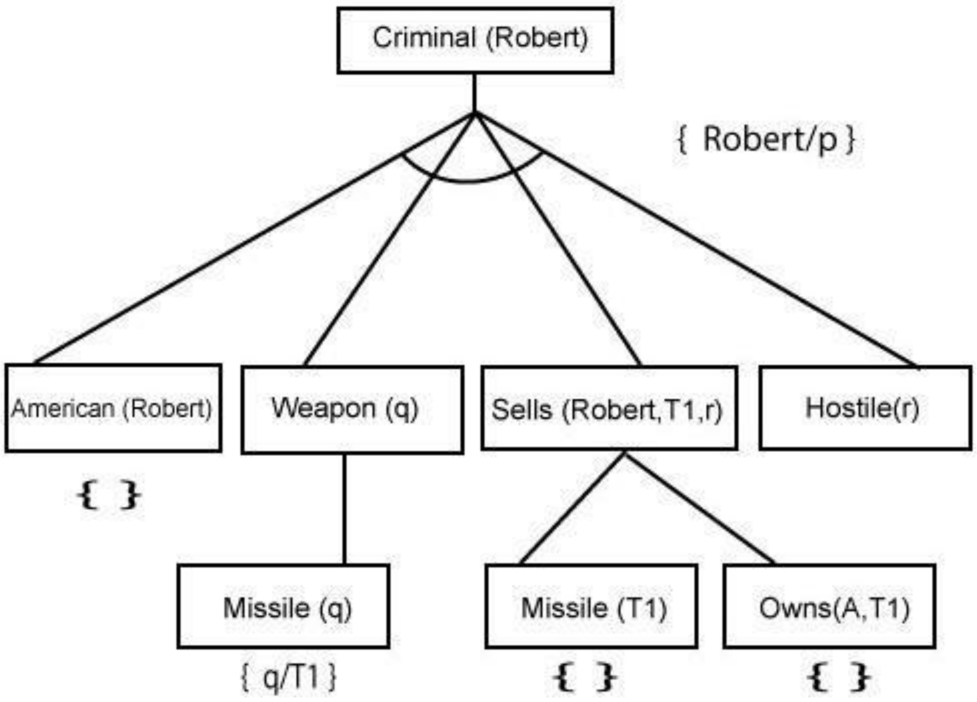


Step-3:t At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q.



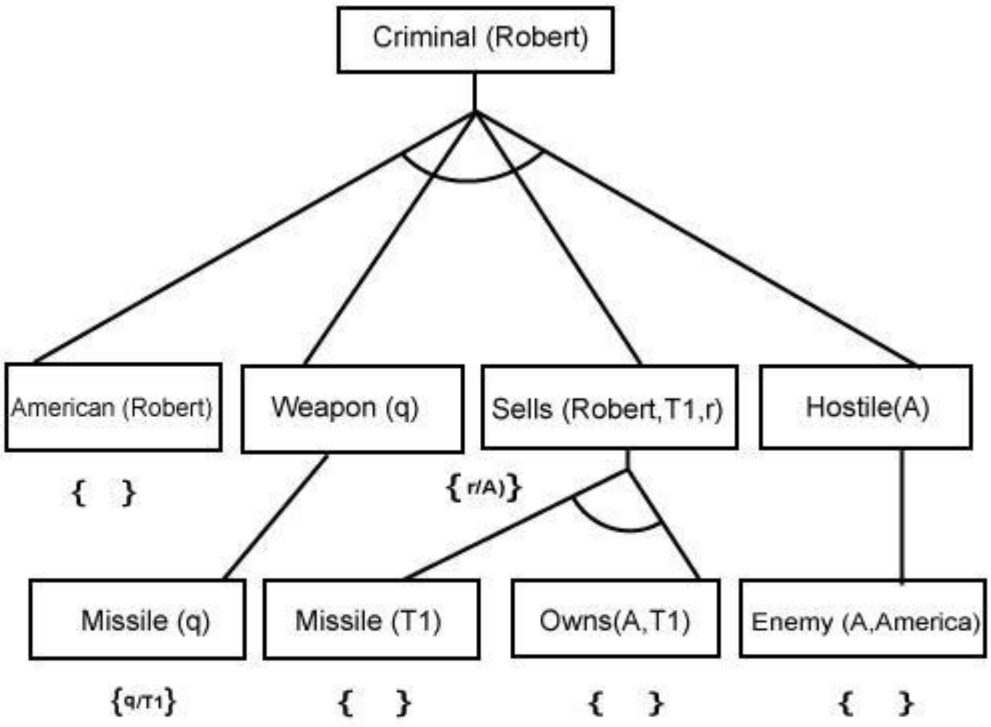
Step-4:

At step-4, we can infer facts Missile(T1) and Owns(A, T1) form Sells(Robert, T1, r) which satisfies the **Rule- 4**, with the substitution of A in place of r. So these two statements are proved here



Step-5:

At step-5, we can infer the fact **Enemy(A, America)** from **Hostile(A)** which satisfies Rule- 6. And hence all the statements are proved true using backward chaining.



Difference between backward chaining and forward chaining

Following is the difference between the forward chaining and backward chaining:

- Forward chaining as the name suggests, start from the known facts and move forward by applying inference rules to extract more data, and it continues until it reaches to the goal, whereas backward chaining starts from the goal, move backward by using inference rules to determine the facts that satisfy the goal.
- Forward chaining is called a **data-driven** inference technique, whereas backward chaining is called a **goal-driven** inference technique.
- Forward chaining is known as the **down-up** approach, whereas backward chaining is known as a **top-down** approach.
- Forward chaining uses **breadth-first search** strategy, whereas backward chaining uses **depth-first search** strategy.
- Forward and backward chaining both applies **Modus ponens** inference rule.
- Forward chaining can be used for tasks such as **planning, design process monitoring, diagnosis, and classification**, whereas backward chaining can be used for **classification and diagnosis tasks**.
- Forward chaining can be like an exhaustive search, whereas backward chaining tries to avoid the unnecessary path of reasoning.
- In forward-chaining there can be various ASK questions from the knowledge base, whereas in backward chaining there can be fewer ASK questions.
- Forward chaining is slow as it checks for all the rules, whereas backward chaining is fast as it checks few required rules only.

S. No.	**Forward Chaining**	**Backward Chaining**
1.	Forward chaining starts from known facts and applies inference rule to extract more data unit it reaches to the goal.	Backward chaining starts from the goal and works backward through inference rules to find the required facts that support the goal.
2.	It is a bottom-up approach	It is a top-down approach
3.	Forward chaining is known as data-driven inference technique as we reach to the goal using the available data.	Backward chaining is known as goal-driven technique as we start from the goal and divide into sub-goal to extract the facts.
4.	Forward chaining reasoning applies a breadth-first search strategy.	Backward chaining reasoning applies a depth-first search strategy.
5.	Forward chaining tests for all the available rules	Backward chaining only tests for few required rules.
6.	Forward chaining is suitable for the planning, monitoring, control, and interpretation application.	Backward chaining is suitable for diagnostic, prescription, and debugging application.
7.	Forward chaining can generate an infinite number of possible conclusions.	Backward chaining generates a finite number of possible conclusions.
8.	It operates in the forward direction.	It operates in the backward direction.
9.	Forward chaining is aimed for any conclusion.	Backward chaining is only aimed for the required data

Resolution in FOL

Resolution

Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e., proofs by contradictions. It was invented by a Mathematician John Alan Robinson in the year 1965.

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**.

Clause: Disjunction of literals (an atomic sentence) is called a **clause**. It is also known as a unit clause.

Conjunctive Normal Form: A sentence represented as a conjunction of clauses is said to be **conjunctive normal form** or **CNF**.

The resolution inference rule:

The resolution rule for first-order logic is simply a lifted version of the propositional rule. Resolution can resolve two clauses if they contain complementary literals, which are assumed to be standardized apart so that they share no variables.

$$\frac{l_1 \lor \ldots\ldots \lor l_k, \qquad m_1 \lor \ldots\ldots \lor m_n}{\text{SUBST}(\theta, l_1 \lor \ldots\ldots \lor l_{i-1} \lor l_{i+1} \lor \ldots \lor l_k \lor m_1 \lor \ldots\ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots\ldots \lor m_n)}$$

Where $\mathbf{l_i}$ and $\mathbf{m_j}$ are complementary literals.

This rule is also called the **binary resolution rule** because it only resolves exactly two literals.

Example:

We can resolve two clauses which are given below:

[Animal (g(x) V Loves (f(x), x)] and [¬ Loves(a, b) V ¬Kills(a, b)]

Where two complimentary literals are: **Loves (f(x), x) and ¬ Loves (a, b)**

These literals can be unified with unifier **θ= [a/f(x), and b/x]** , and it will generate a resolvent clause:

[Animal (g(x) V ¬ Kills(f(x), x)].

Steps for Resolution:

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

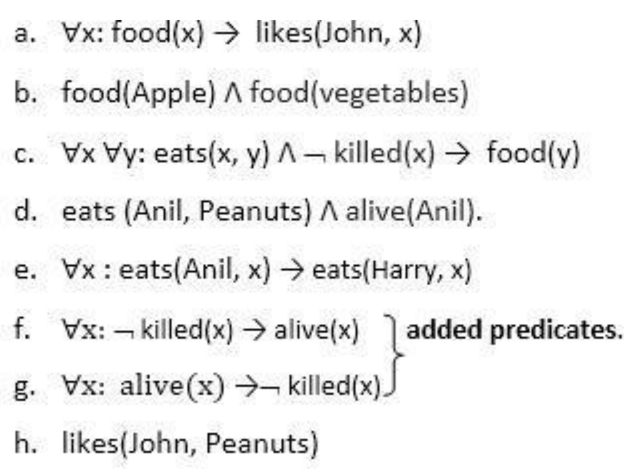
To better understand all the above steps, we will take an example in which we will apply resolution.

Example:

1. **John likes all kind of food.**
2. **Apple and vegetable are food**
3. **Anything anyone eats and not killed is food.**
4. **Anil eats peanuts and still alive**
5. **Harry eats everything that Anil eats.**
   Prove by resolution that:
6. **John likes peanuts.**

Step-1: Conversion of Facts into FOL

In the first step we will convert all the given statements into its first order logic.



a. ∀x: food(x) → likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀x ∀y: eats(x, y) Λ ¬ killed(x) → food(y)

d. eats (Anil, Peanuts) Λ alive(Anil).

e. ∀x : eats(Anil, x) → eats(Harry, x)

f. ∀x: ¬ killed(x) → alive(x)

g. ∀x: alive(x) →¬ killed(x)

added predicates.

h. likes(John, Peanuts)

Step-2: Conversion of FOL into CNF

In First order logic resolution, it is required to convert the FOL into CNF as CNF form makes easier for resolution proofs.

- **Eliminate all implication (→) and rewrite**
   1. ∀x ¬ food(x) V likes(John, x)

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

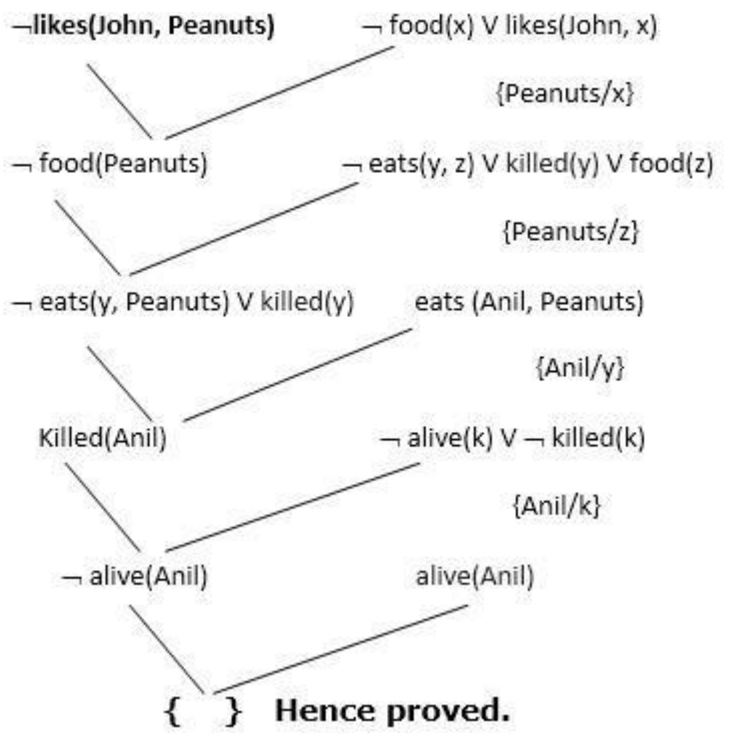
- **Move negation (¬)inwards and rewrite**
    1. ∀x ¬ food(x) V likes(John, x)
    2. food(Apple) Λ food(vegetables)
    3. ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)
    4. eats (Anil, Peanuts) Λ alive(Anil)
    5. ∀x ¬ eats(Anil, x) V eats(Harry, x)
    6. ∀x ¬killed(x)] V alive(x)
    7. ∀x ¬ alive(x) V ¬ killed(x)
    8. likes(John, Peanuts).
- **Rename variables or standardize variables**
    1. ∀x ¬ food(x) V likes(John, x)
    2. food(Apple) Λ food(vegetables)
    3. ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)
    4. eats (Anil, Peanuts) Λ alive(Anil)
    5. ∀w¬ eats(Anil, w) V eats(Harry, w)
    6. ∀g ¬killed(g)] V alive(g)
    7. ∀k ¬ alive(k) V ¬ killed(k)
    8. likes(John, Peanuts).
- **Eliminate existential instantiation quantifier by elimination.**
  In this step, we will eliminate existential quantifier ∃, and this process is known as **Skolemization**. But in this example problem since there is no existential quantifier so all the statements will remain same in this step.
- **Drop Universal quantifiers.**
  In this step we will drop all universal quantifier since all the statements are not implicitly quantified so we don't need it.
    1. ¬ food(x) V likes(John, x)
    2. food(Apple)
    3. food(vegetables)
    4. ¬ eats(y, z) V killed(y) V food(z)
    5. eats (Anil, Peanuts)
    6. alive(Anil)
    7. ¬ eats(Anil, w) V eats(Harry, w)
    8. killed(g) V alive(g)
    9. ¬ alive(k) V ¬ killed(k)
    10. likes(John, Peanuts).
- **Distribute conjunction ∧ over disjunction ¬.**
  This step will not make any change in this problem.

Step-3: Negate the statement to be proved

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

Step-4: Draw Resolution graph:

Now in this step, we will solve the problem by resolution tree using substitution. For the above problem, it will be given as follows:



Hence the negation of the conclusion has been proved as a complete contradiction with the given set of statements.

Explanation of Resolution graph:

- In the first step of resolution graph, **¬likes(John, Peanuts)** , and **likes(John, x)** get resolved(canceled) by substitution of **{Peanuts/x}**, and we are left with **¬ food(Peanuts)**
- In the second step of the resolution graph, ¬ **food(Peanuts)** , and **food(z)** get resolved (canceled) by substitution of **{ Peanuts/z}**, and we are left with ¬ **eats(y, Peanuts) V killed(y)** .
- In the third step of the resolution graph, ¬ **eats(y, Peanuts)** and **eats (Anil, Peanuts)** get resolved by substitution **{Anil/y}**, and we are left with **Killed(Anil)** .
- In the fourth step of the resolution graph, **Killed(Anil)** and ¬ **killed(k)** get resolve by substitution **{Anil/k}**, and we are left with ¬ **alive(Anil)** .
- In the last step of the resolution graph ¬ **alive(Anil)** and **alive(Anil)** get resolved.

What is Bayes' Theorem?

Bayes theorem (also known as the Bayes Rule or Bayes Law) is used to determine the conditional probability of event A when event B has already occurred.

The general statement of Bayes' theorem is "**The conditional probability of an event A, given the occurrence of another event B, is equal to the product of the event of B, given A and the probability of A divided by the probability of event B.**" i.e

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

where,

- **$P(A)$** and **$P(B)$** are the probabilities of events A and B
- **$P(A|B)$** is the probability of event A when event B happens
- **$P(B|A)$** is the probability of event B when A happens

Bayes Theorem Statement

Bayes' Theorem for n set of events is defined as,

Let $E_1, E_2,\dots, E_n$ be a set of events associated with the sample space S, in which all the events $E_1, E_2,\dots, E_n$ have a non-zero probability of occurrence. All the events $E_1, E_2,\dots,$ E form a partition of S. Let A be an event from space S for which we have to find probability, then according to Bayes' theorem,

$P(E_i|A) = P(E_i)P(A|E_i) / \sum P(E_k)P(A|E_k)$

for $k = 1, 2, 3, \dots., n$

Terms Related to Bayes Theorem

In Bayes' theorem, the likelihood is one of the key components that helps update the probability of a hypothesis given new evidence. It refers to the probability of observing the given data under a specific hypothesis.

Conditional Probability

- The probability of an event A based on the occurrence of another event B is termed conditional Probability.
- It is denoted as **$P(A|B)$** and represents the probability of A when event B has already happened.

Joint Probability

When the probability of two more events occurring together and at the same time is measured it is marked as Joint Probability. For two events A and B, it is denoted by joint probability is denoted as, **$P(A\cap B)$.**

Random Variables

Real-valued variables whose possible values are determined by random experiments are called random variables. The probability of finding such variables is the experimental probability.

Bayes’ Theorem Applications

Bayesian inference is very important and has found application in various activities, including medicine, science, philosophy, engineering, sports, law, etc., and Bayesian inference is directly derived from Bayes’ theorem.

Example: Bayes’ theorem defines the accuracy of the medical test by taking into account how likely a person is to have a disease and what is the overall accuracy of the test.

Bayes Theorem Examples

Example 1: A person has undertaken a job. The probabilities of completion of the job on time with and without rain are 0.44 and 0.95 respectively. If the probability that it will rain is 0.45, then determine the probability that the job will be completed on time.

Let E_1 be the event that the mining job will be completed on time and E_2 be the event that it rains. We have,

$P(A) = 0.45$,
$P(\text{no rain}) = P(B) = 1 − P(A) = 1 − 0.45 = 0.55$

By multiplication law of probability,

$P(E_1) = 0.44$, and $P(E_2) = 0.95$

Since, events A and B form partitions of the sample space S, by total probability theorem, we have

$P(E) = P(A)\, P(E_1) + P(B)\, P(E_2)$

$\sim P(E) = 0.45 \times 0.44 + 0.55 \times 0.95$

$\sim P(E) = 0.198 + 0.5225 = 0.7205$

So, the probability that the job will be completed on time is 0.7205

Example : 2
A Factory has 2 machines producing widgets. Machine A produces 60% of the widgets and has a defect rate of 2%. Machine B produces 40% of the widgets and has a defect rate of 3%. If a randomly selected widget is found to be defective, what is the probability that it was produced by Machine A?[Hint: Use Bayes Theorem]

Step 1: Define Events

- Let A be the event that a widget is produced by **Machine A**.
- Let B be the event that a widget is produced by **Machine B**.
- Let D be the event that a widget is **defective**.

given:

P(A)=0.6,P(B)=0.4P(A) = 0.6,

P(D|A)=0.02,P(D|B)=0.03

We need to find P(A|D), the probability that a defective widget was produced by Machine A.

Step 2: Apply Bayes' Theorem

Bayes' theorem states:

$$P(A|D) = \frac{P(D|A)P(A)}{P(D)}$$

where P(D), the total probability of a defective widget, is found using the **Law of Total Probability**:

P(D)=P(D|A)P(A)+P(D|B)P(B)

Step 3: Compute P(D)

P(D)=(0.02×0.6)+(0.03×0.4)P

0.024=0.012+0.012=0.024

Step 4: Compute P(A|D)

$$P(A|D) = \frac{(0.02 \times 0.6)}{0.024}$$

$$= \frac{0.012}{0.024} = 0.5$$

Final Answer:

P(A|D)=0.5

Thus, if a randomly selected widget is found to be defective, there is a **50% probability that it was produced by Machine A**.