**What is Propositional Logic in AI?**

This logic is synonymous with sentential logic, provides ways and means of combining or changing the propositions to create a complicated structure and to build a new logical relationship and properties. It combines logical connections of all the constituent statements, and the true value of the complex statement is derived. While doing, it considers other factors like reasoning, relationship, and interconnection between the constituent statements.

This logic provides better clarity on data and information in an incomplete environment by deeper analysis and inference of the limited information presented to it. Due to this unique feature and powerful algorithms deployed in logical reasoning, it is widely adopted in advanced search in AI in solving complex problems.

**Syntax**

Propositional logic (PL), in order to be effective, then we need to follow a language structure that should be agreed upon by everyone, and it should be easy to adopt by all. PL Language structure consists of simple undividable statements joined together with logical connectors.

|  |  |  |
| --- | --- | --- |
| **Sr.No** | **Subject** | **Syntax** |
| 1 | Simple undividable statement represent true or false (not both) and it is Boolean in nature | Upper Case letters A, B, C, P, Q, R are used to represent statements |
| 2 | Logical Connectors or operators used to connect two statements | ^, v, →, ↔, ¬ are used to represent AND, OR, Implies, bi-conditional and NOT condition. |
| 3 | Complex conditions | Complex conditions are handled by coding connectors within parenthesis. |

* A simple sentence is called Atomic Proposition, and it should be either true or false.

**Example**: 9+2 =11 is one such proposition, and it is true.

Sunrises in the west is another example, and it is false.

* A combination of simple sentences connected by logical connectors is called Compound.

**Example**: Today is Friday and people visit Temple today. It’s raining, and

the match is called off.

* A Proposition that is always true is known as **Tautology** (another name for Valid Sentence)
* A Proposition that is always false is known as **Contradiction.**
* Sentences that are questions and command in nature do not belong to this Proposition Category.

**Logical Connectives**

It connects two undividable simple sentences or expresses a sentence in a logical sense. Complex statements can be created using logical connectives. There are 5 types of connectors, namely.

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.** | **Type** | **Symbol** | **Description** |
| 1 | Negation | ¬ P | It represents a Negative condition. P is a positive statement, and ¬ P indicates NOT condition.  **Example:** Today is Monday (P), Today is not a Monday (¬ P) |
| 2 | Conjunction | P ^ Q | It joins two statements P, Q with AND clause.  **Example:** Ram is a cricket player (P). Ram is a Hockey player (Q). Ram plays both cricket and Hockey is represented by (P ^ Q) |
| 3 | Disjunction | P v Q | It joins two statements P, Q with OR Clause.  **Example:** Ram leaves for Mumbai (P) and Ram leaves for Chennai (Q).  **Ram leaves for Chennai or Mumbai is represented by (P v Q).** In this complex statement, at any given point of time if P is True Q is not true and vice versa. |
| 4 | Implication | P → Q | Sentence (Q) is dependent on sentence (P), and it is called implication. It follows the rule of If then clause. If sentence P is true, then sentence Q is true. The condition is unidirectional.  **Example:** If it is Sunday (P) then I will go to Movie (Q), and it is represented as P → Q |
| 5 | Bi-conditional | P **⇔**Q | Sentence (Q) is dependent on sentence (P), and vice versa and conditions are bi-directional in this connective. If a conditional statement and its converse are true, then it is called as bi-conditional connective (Implication condition in both the directions P → Q and Q → P). If and only if all conditions are true, then the end statement is true.  **Example:** If I have 1000 Rupees then only I will go to Bar.  The converse condition that I will go to Bar if and only if I have Rs 1000. The first statement covers necessity and the second one covers sufficiency. |

The following table depicts the truth values of various combinations of Boolean conditions for Statements P and Q for all the logical connectives.

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

This can be extended to for three statements (P, Q, R) with any combinations of logical connectives. The connectives can be combined with multiple connectors like Parenthesis or Brackets. The order of precedence of logical connectors in evaluating propositional logic is

1. Parenthesis
2. Negation
3. Conjunction (AND)
4. Disjunction (OR)
5. Implication (If…then)
6. Bi-Conditional (if and only if)

ADVANTAGE: Due to PL ability in solving complex problems this logic is used quite extensively in Business, Education and Medical fields.

But it has some limitations

1. It cannot address relations like Some, ALL,
2. It can neither handle logical relationships.
3. It has limited expressive ability.

|  |  |  |
| --- | --- | --- |
| **Laws of Propositional Logic List** | | |
| De Morgan’s laws: | ¬( p ∨ q ) ≡ ¬p ∧ ¬q | ¬( p ∧ q ) ≡ ¬p ∨ ¬q |
| Idempotent laws: | p ∨ p ≡ p | p ∧ p ≡ p |
| Associative laws: | ( p ∨ q ) ∨ r ≡ p ∨ ( q ∨ r ) | ( p ∧ q ) ∧ r ≡ p ∧ ( q ∧ r ) |
| Commutative laws: | p ∨ q ≡ q ∨ p | p ∧ q ≡ q ∧ p |
| Distributive laws: | p ∨ ( q ∧ r ) ≡ ( p ∨ q ) ∧ ( p ∨ r ) | p ∧ ( q ∨ r ) ≡ ( p ∧ q ) ∨ ( p ∧ r ) |
| Double negation law: | ¬¬p ≡ p |  |
| Absorption laws: | p ∨ (p ∧ q) ≡ p | p ∧ (p ∨ q) ≡ p |
| Law of contradiction: | p∧ ¬p =false (0) |  |
| Law of excluded middle | p ∨¬p=true (1) |  |

**RULES OF INFERENCE:**

**Modus Ponens** **Modus Tollens**

P: It is bright and sunny today. P: I will not wear my sunglasses.

Q: Therefore, I will wear my sunglasses. Q: Therefore, it is not bright and sunny today.

which states that if ***p*** is true, and if (***pimpliesq***), If ***q*** is false, and if (***phttps://www.csm.ornl.gov/~sheldon/ds/images/1logde.gifq***), then ***p*** is also false

then ***q*** is true

NOTATION: NOTATION:

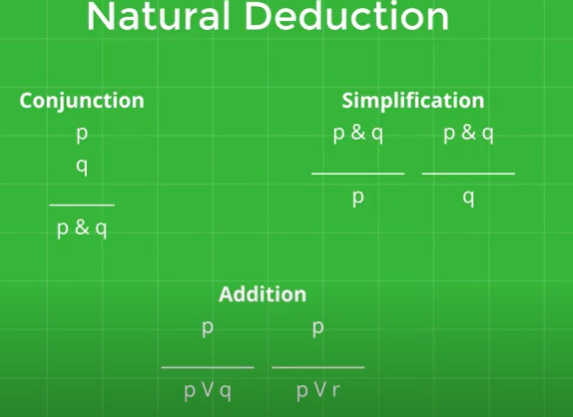
P Q, P P Q, ~Q

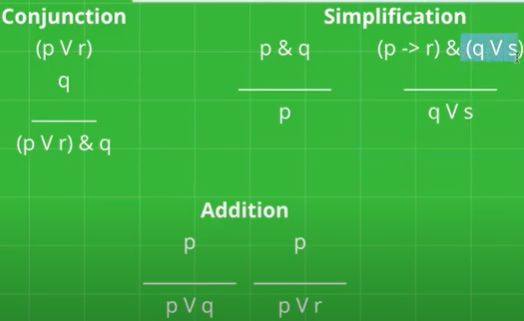
https://www.csm.ornl.gov/~sheldon/ds/images/tf.gif Q https://www.csm.ornl.gov/~sheldon/ds/images/tf.gif ~P

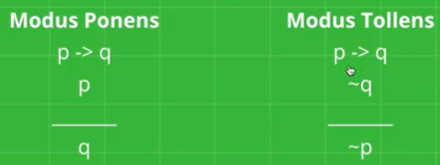
**D E F I N I T I O N S**

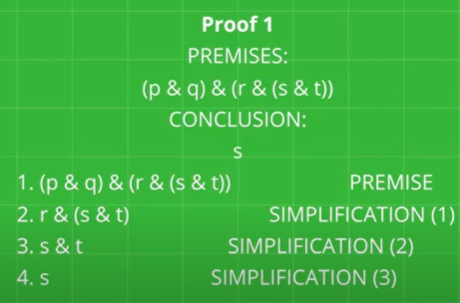
|  |  |
| --- | --- |
| Argument | A series of statements . |
| Premises | All statements in an argument except the final one. |
| Conclusion | The final (or concluding) statement in an argument. |
| https://www.csm.ornl.gov/~sheldon/ds/images/tf.gif | Symbol for "therefore", normally used to identify the conclusion of an argument. |
| Modus Ponens | "method of affirming." A rule of inference used to draw logical conclusions, which states that if ***p*** is true, and if ***p*** implies ***q*** (***pimpliesq***), then ***q*** is true. |
| Modus Tollens | "method of denying." A rule of inference drawn from the combination of modus ponens and the contrapositive. If ***q*** is false, and if ***p*** implies ***q*** (***phttps://www.csm.ornl.gov/~sheldon/ds/images/1logde.gifq***), then ***p*** is also false. |
| Fallacy | An error in reasoning. |

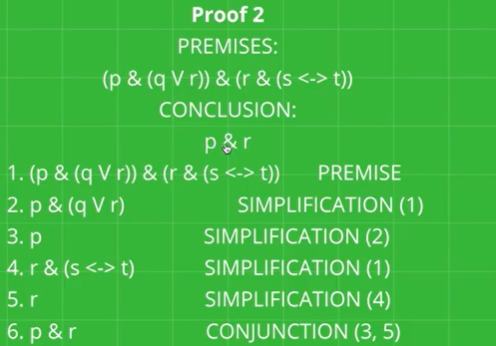
NATURAL DEDUCTION:

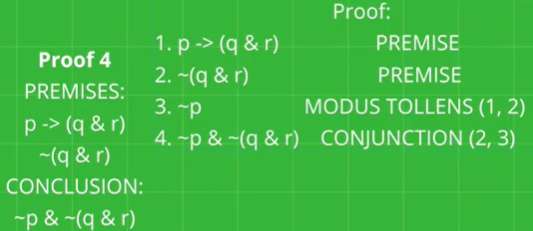


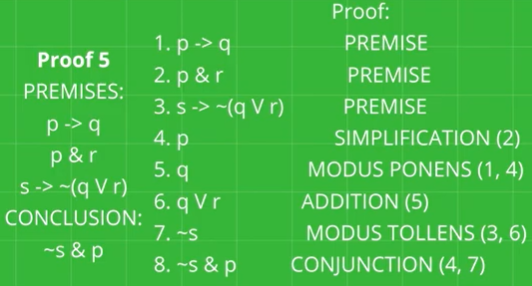












RULES FOR DOUBLE NEGATION:

~~P ~~e [ELEMINATION OF ~~]

P resoli

P ~~i [INRODUCTION OF ~~]

~~P

EXAMPLE:re

P ,~~(Q^R) I------ ~~P^R

1.P PREMISE

2.~~(Q^R) PREMISE

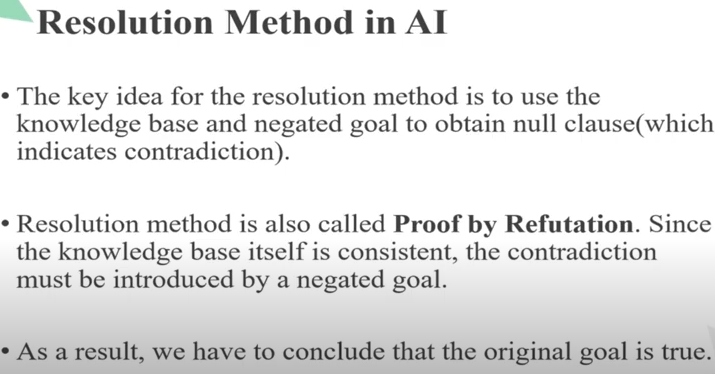
3.~~P ~~I 1

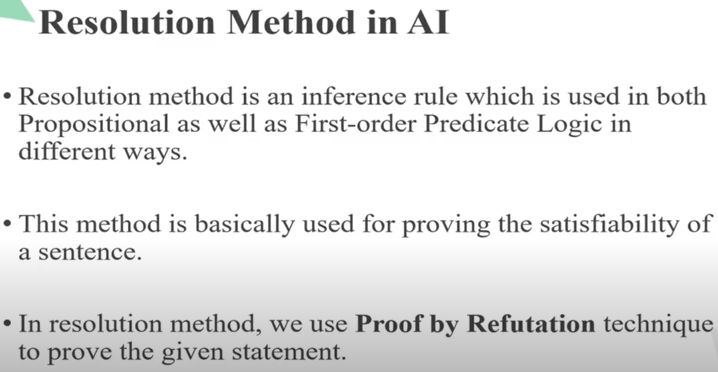
4.Q^R ~~e 2

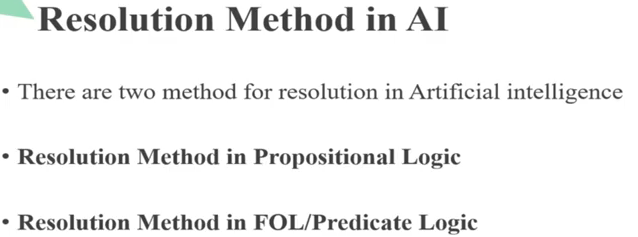
5.R ^e 4

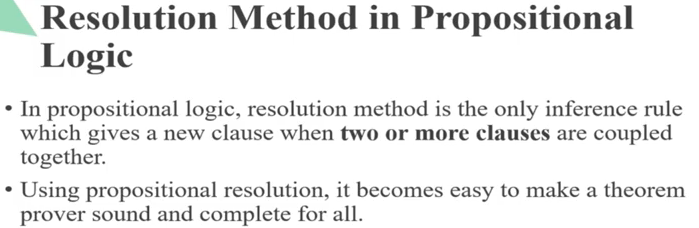
6 ~~P^R ^I (3,4)

RESOLUTION REFUTATION IN PROPOSITIONAL LOGIC:

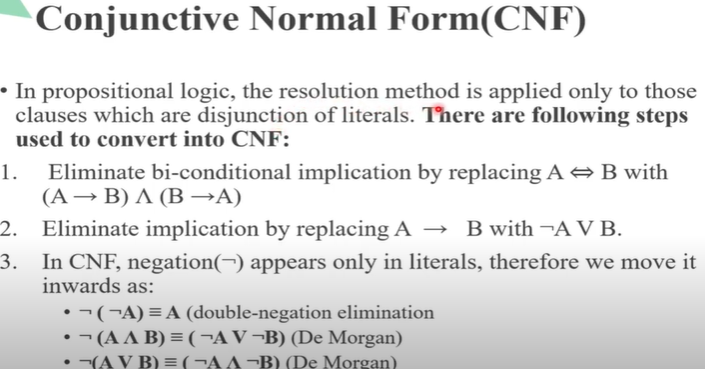


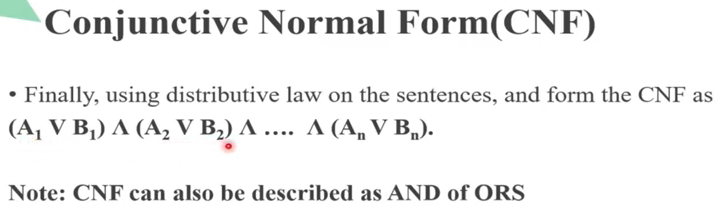


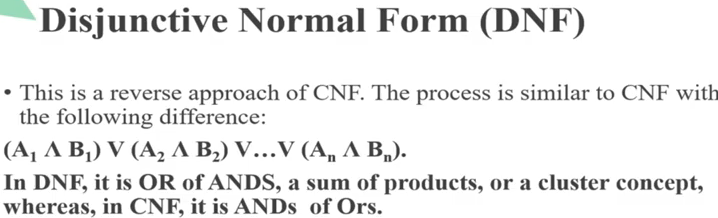


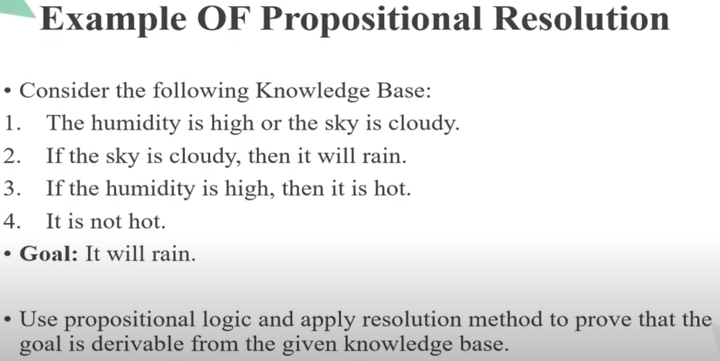


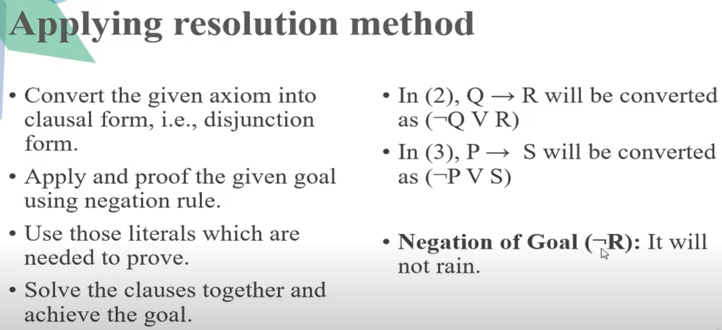










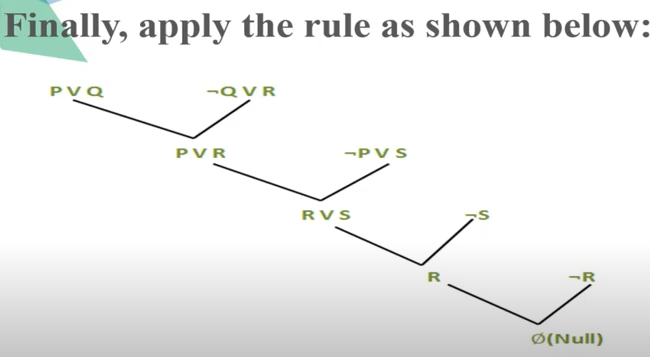


HUMIDITY IS HIGH (P) OR SKY IS CLOUDY (Q) : P v Q

IF THE SKY IS CLOUDY(Q),THEN IT WILL RAIN(R) : Q->R----------EQ(1) FROM EQ(1) ~Q v R

IF THE HUMIDITY IS HIGH(P),THEN IT IS HOT(S) : P->S-------------EQ(2) FROM EQ(2) ~P v S

IT IS NOT HOT : ~S



EXAMPLE (2):

GIVEN AXIOMS: CONVERTING TO CLAUSE FORM:

P P

(P^Q)->R ~(P^Q)vR = ~Pv~QvR

(SvT)->Q ~(SvT)vQ = (~S ^~T) v Q = (~SvQ) ^ (~TvQ) =(~SvQ) , (~TvQ)

T T

**TO PROVE :R**

~Pv~QvR **~R**

~Pv~Q P

~Q ~TvQ

**~T T**

NULL

First-Order Logic

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

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

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

**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......
  + **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:

59.1M

967

Hello Java Program for Beginners

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

Atomic sentences:

* Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.
* We can represent atomic sentences as **Predicate (term1, term2, ......, term n)**.

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

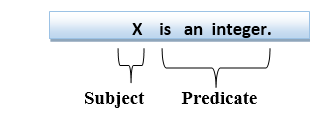
Complex Sentences:

* Complex sentences are made by combining atomic sentences using connectives.

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

* **Subject:** Subject is the main part of the statement.
* **Predicate:** A predicate can be defined as a relation, which binds two atoms together in a statement.

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



Quantifiers in First-order logic:

* A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
* These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifier:
  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 "→".

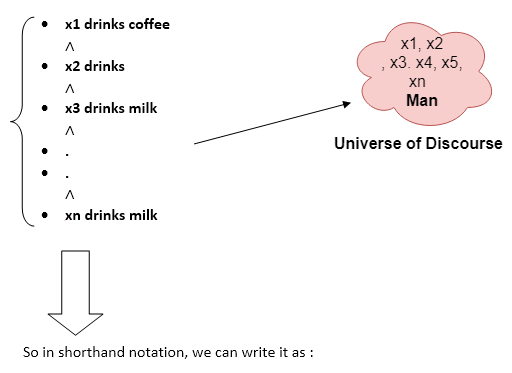
If x is a variable, then ∀x is read as:

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

Example:

**All man drink coffee.**

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



**∀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.

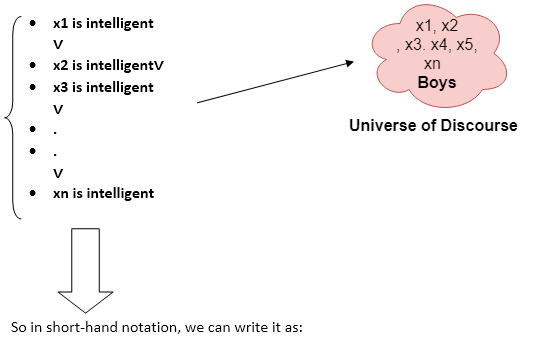
Note: In Existential quantifier we always use AND or Conjunction symbol (∧).

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

* **There exists a 'x.'**
* **For some 'x.'**
* **For at least one 'x.'**

Example:

**Some boys are intelligent.**



**∃x: boys(x) ∧ intelligent(x)**

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

Points to remember:

* The main connective for universal quantifier **∀** is implication **→**.
* The main connective for existential quantifier **∃** is and **∧**.

**Properties of Quantifiers**:

* In universal quantifier, ∀x∀y is similar to ∀y∀x.
* In Existential quantifier, ∃x∃y is similar to ∃y∃x.
* ∃x∀y is not similar to ∀y∃x.

Some Examples of FOL using quantifier:

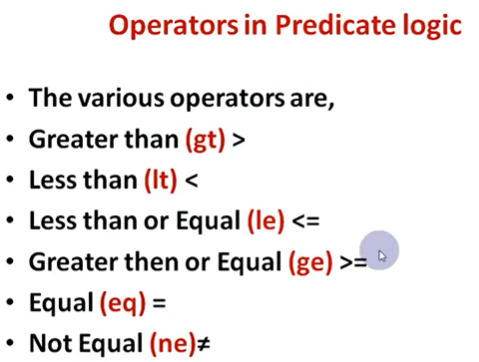
**1. All birds fly.**  
In this question the predicate is "**fly(bird)**."  
And since there are all birds who fly so it will be represented as follows.  
              **∀x: bird(x) →fly(x)**.

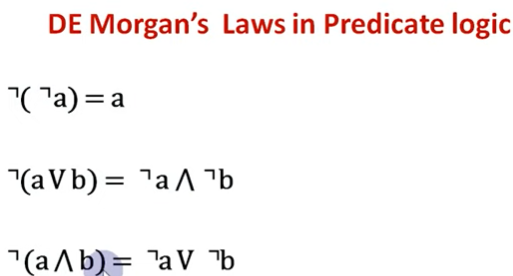
**2. Every man respects his parent.**  
In this question, the predicate is "**respect(x, y)," where x=man, and y= parent**.  
Since there is every man so will use ∀, and it will be represented as follows:  
              **∀x: man(x) → respects (x, parent)**.

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

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

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

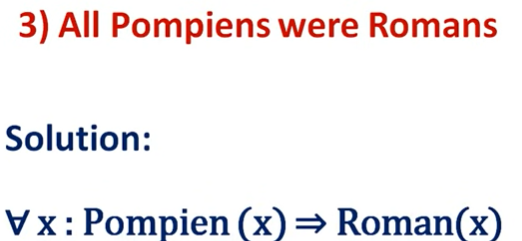


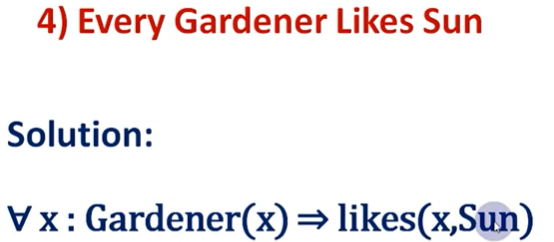


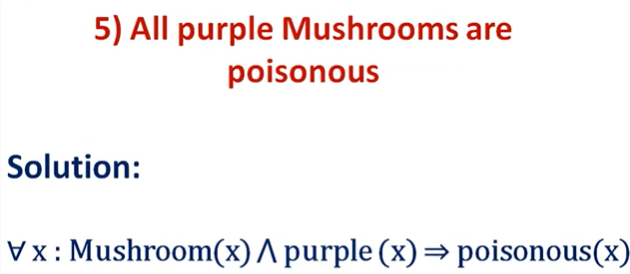
EXAMPLES:

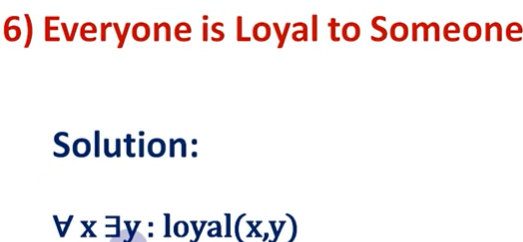
Marcus is a man

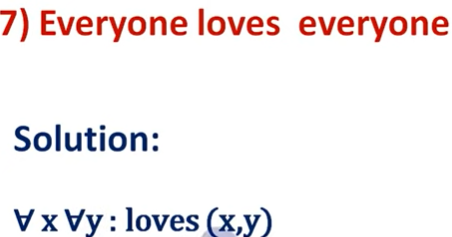
Man(marcus)

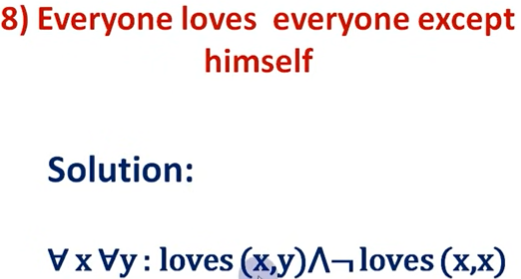


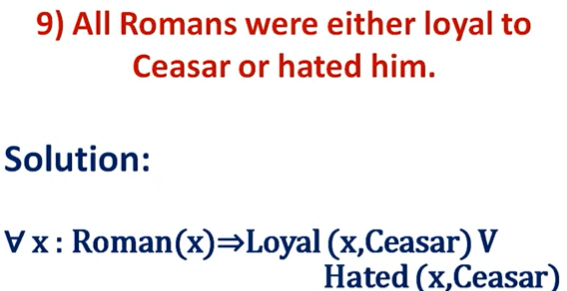


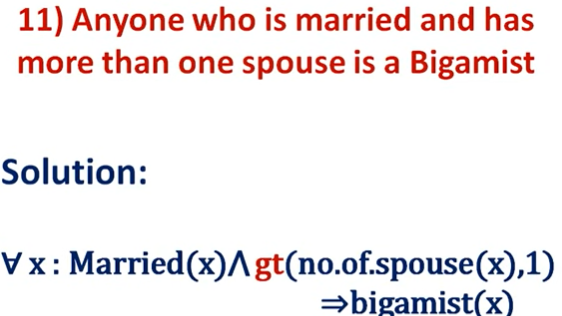


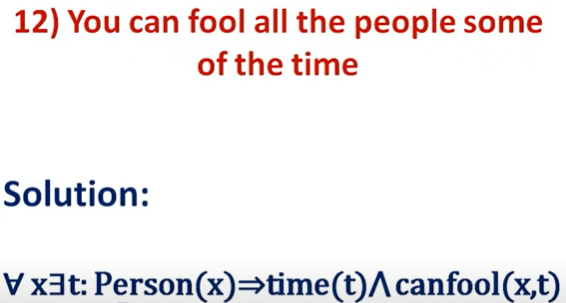


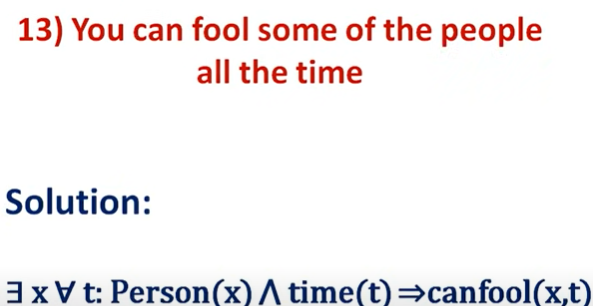


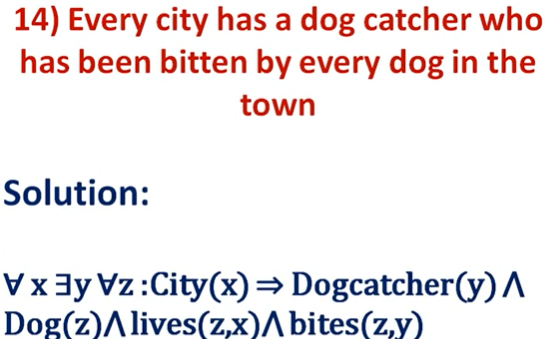












## Representing Knowledge using rules in AI

## C:\Users\user\Pictures\Screenshots\Screenshot (108).png

## C:\Users\user\Pictures\Screenshots\Screenshot (109).png

## C:\Users\user\Pictures\Screenshots\Screenshot (110).png

## C:\Users\user\Pictures\Screenshots\Screenshot (111).png

## C:\Users\user\Pictures\Screenshots\Screenshot (107).png

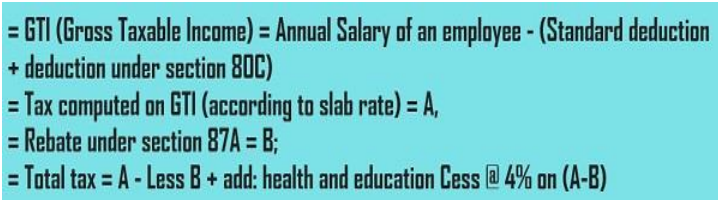
**Procedural and Declarative Knowledge :**

1**. Procedural Knowledge** :

• The Procedural knowledge is a type of knowledge where the essential control information that is required is integrated in the knowledge itself.

• It also used with an interpreter to employ the knowledge which follows the instructions given in the knowledge.

**Ex -** It can include a group of logical assertions merged with a resolution theorem prove to provide an absolute program for solving problems. Here, the implied income tax of an employee salary can be thought of as a procedural knowledge as it would require a process to calculate it as given below

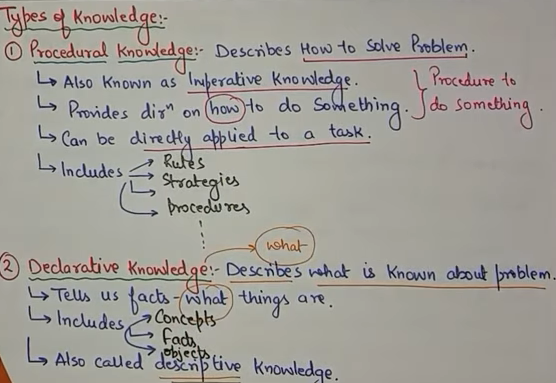


• So, this is how the tax of an employee is calculated by following a lengthy process instead of just collecting facts.

2. **Declarative Knowledge**

• A Declarative knowledge is where only knowledge is described but not the use to which the knowledge is employed is not provided.

• So, in order to use this declarative knowledge, we need to add it with a program that indicates what is to be done to the knowledge and how it is to be done



Difference the Procedural and Declarative Knowledge

PROCEDURAL KNOWLEDGE DECLARATIVE KNOWLEDGE

It is also known as Interpretive It is also known as Descriptive

knowledge knowledge

Procedural Knowledge means how a While Declarative Knowledge

particular thing can be accomplished means basic knowledge about

something

Procedural Knowledge is generally not Declarative Knowledge is more popular. popular

Procedural Knowledge can’t be easily Declarative Knowledge can be

communicate. Easily communicate

Procedural Knowledge is generally Declarative Knowledge is data

process oriented in nature oriented in nature

In Procedural Knowledge debugging In Declarative Knowledge debugging

and validation is not easy. and validation is easy

**Forward Reasoning**

• The solution of a problem generally includes the initial data and facts in order to arrive at the solution. These unknown facts and information is used to deduce the result

• For example, while diagnosing a patient ,the doctor first check the symptoms and medical condition of the body such as temperature, blood pressure, pulse, eye colour, blood, etcetera. After that, the patient symptoms are analysed and compared against the predetermined symptoms. Then the doctor is able to provide the medicines according to the symptoms of the patient. So, when a solution employs this manner of reasoning, it is known as forward reasoning

Steps that are followed in the forward reasoning :

**1.** In the first step, the system is given one or more than one constraints.

**2**. Then the rules are searched in the knowledge base for each constraint. The rules that fulfill the condition are selected(i.e., IF part).

**3.** Now each rule is able to produce new conditions from the conclusion of the invoked one. As a result, THEN part is again included in the existing one.

**4.** The added conditions are processed again by repeating step 2. The process will end if there is no new conditions exist

For example, consider the following set of rules that is used to control an elevator in a three-stair building:

**Rule 1**

IF on first floor and button is pressed on first floor

THEN open door

**Rule 2**

  IF on first floor AND button is pressed on second floor

 THEN go to second floor

**Rule 3**

 IF on first floor AND button is pressed on third floor

   THEN go to third floor

**Rule 4**

  IF on second floor AND button is pressed on first floor AND already going to third floor

  THEN remember to go to first floor later

This represents just a subset of the rules that would be needed, but we can use it to illustrate how forward chaining works.

Let us imagine that we start with the following facts in our database:

**Fact 1**

  At first floor

**Fact 2**

  Button pressed on third floor

**Fact 3**

Today is Tuesday

Now the system examines the rules and finds that Facts 1 and 2 match the antecedents (logically precedes) of Rule 3. Hence, Rule 3 fires, and its conclusion “Go to third floor” is added to the database of facts. Presumably, this results in the elevator heading toward the third floor.

Note that Fact 3 was ignored altogether because it did not match the antecedents of any of the rules.

Now let us imagine that the elevator is on its way to the third floor and has reached the second floor, when the button is pressed on the first floor. The fact Button pressed on first floor

Is now added to the database, which results in Rule 4 firing.

Now let us imagine that later in the day the facts database contains the following information:

**Fact 1**

 At first floor

**Fact 2**

 Button pressed on second floor

**Fact 3**

 Button pressed on third floor

In this case, two rules are triggered—Rules 2 and 3. In such cases where there is more than one possible conclusion, **conflict resolution** needs to be applied to decide which rule to fire.

**Conflict Resolution**

In a situation where more than one conclusion can be deduced from a set of facts, there are a number of possible ways to decide which rule to fire.

For example, consider the following set of rules:

 IF it is cold

  THEN wear a coat

IF it is cold

THEN stay at home

 IF it is cold

  THEN turn on the heat

If there is a single fact in the fact database, which is “it is cold,” then clearly there are three conclusions that can be derived. In some cases, it might be fine to follow all three conclusions, but in many cases the conclusions are incompatible.

In one conflict resolution method, rules are given priority levels, and when a conflict occurs, the rule that has the highest priority is fired, as in the following example:

IF patient has pain

  THEN prescribe painkillers priority 10

 IF patient has chest pain

  THEN treat for heart disease priority 100

**Backward Reasoning** :

• The backward reasoning is inverse of forward reasoning in which goal is analyzed in order to deduce the rules, initial facts and data.

• We can understand the concept by the similar example given in the above definition, where the doctor is trying to diagnose the patient with the help of the inceptive data such as symptoms. However, in this case, the patient is experiencing a problem in his body, on the basis of which the doctor is going to prove the symptoms. This kind of reasoning comes under backward reasoning

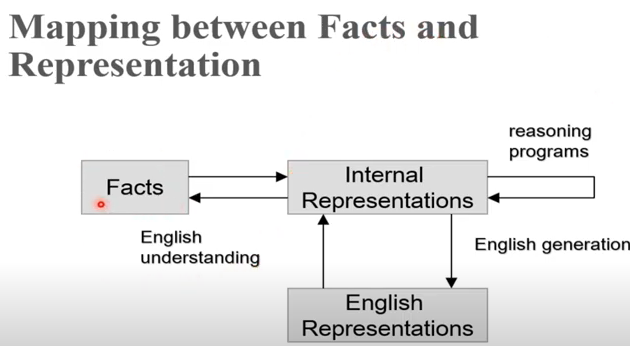
Steps that are followed in the backward reasoning :

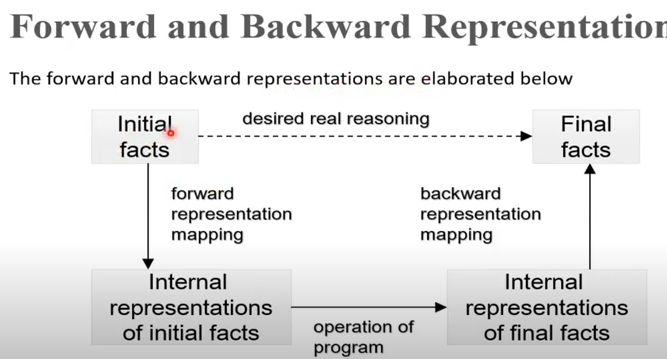
**1.** Firstly, the goal state and the rules are selected where the goal state reside in the THEN part as the conclusion.

**2.** From the IF part of the selected rule the sub goals are made to be satisfied for the goal state to be true.

**3.** Set initial conditions important to satisfy all the sub goals.

**4.** Verify whether the provided initial state matches with the established states. If it fulfills the condition then the goal is the solution otherwise other goal state is selected

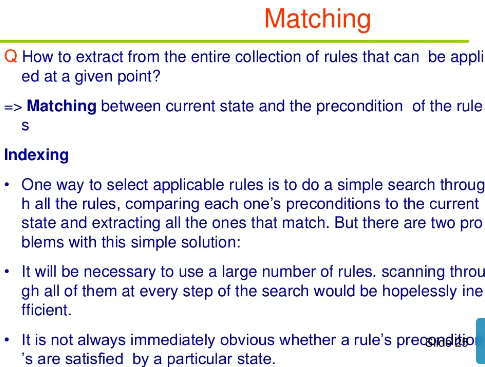


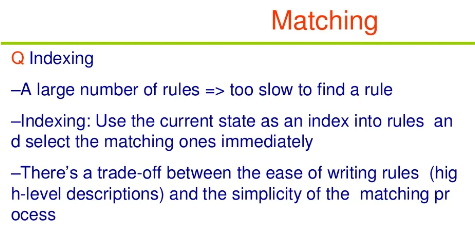


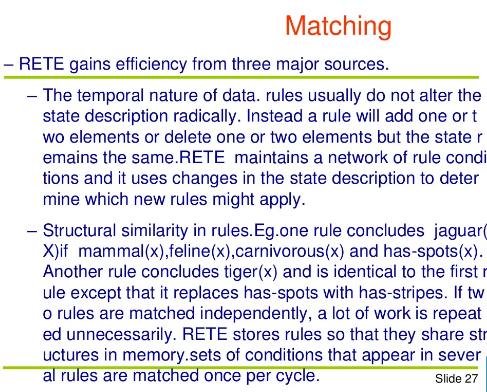
Difference between backward chaining and forward chaining

Forward Chaining Backward Chaining

|  |  |
| --- | --- |
| It starts from known facts and  applies inference rule to extract  more data until it reaches to the  goal. | It starts from the goal and works  backward through inference  rules to find the required facts  that support the goal. |
| It is a bottom-up approach | It is a top-down approach |
| It is known as data-driven  inference technique as we reach  to the goal using the available  data. | It is known as goal-driven  technique as we start from the  goal and divide into sub-goal to  extract the facts. |
| It tests for all the available rules | It tests only for few required  rules. |
| It is suitable for the planning,  monitoring, control, and  interpretation application. | It is suitable for diagnostic,  prescription, and debugging  application. |
| It can generate an infinite  number of possible conclusions. | It generates a finite number of  possible conclusions. |
| It operates in the forward  direction. | It operates in the backward  direction. |
| It applies a breadth-first search  strategy. | It applies a depth-first search  Strategy |



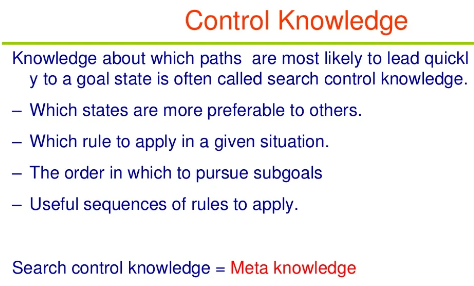


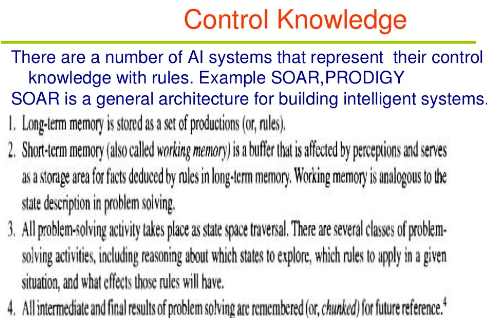


The Rete algorithm is a pattern matching algorithm for implementing rule-based systems. The algorithm was developed to efficiently apply many rules or patterns to many objects, or facts, in a knowledge base. It is used to determine which of the system's rules should fire based on its data store, its facts







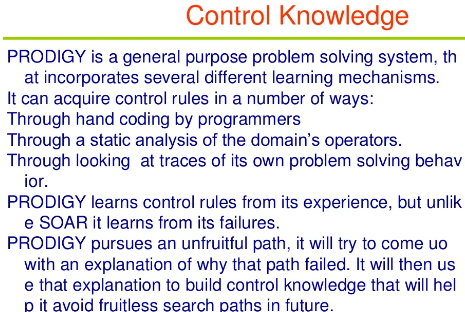


SOAR ARCHITECTURE:



Soar is **a general cognitive architecture for developing systems that exhibit intelligent behavior**

Prodigy is **a modern annotation tool for creating training and evaluation data for machine learning models**



for

