ARTIFICIAL INTELLIGENCE

UNIT-II LOGICAL

REASONING

Logical Agents – propositional logic – inferences – first-order logic – inferences in first-order logic – forward chaining- backward chaining – unification – resolution

2.0 Logic:

- A knowledge representation language in which syntax and semantics are defined correctly is known as logic.
- A formal language to represent the knowledge in which reasoning is carried out to achieve the goal state.
- Logics consists of the following two representations in sequence,
  - ➢ A formal system is used to describe the state of the world
    - ✓ Syntax " Which describes how to make sentences"
    - ✓ Semantics " Which describes the meaning of the sentences"
  - ➢ The proof theory "a set of rule for deducing the entailments of a set of sentences.
- We will represent the sentences using two different logics, They are,
  - ➢ Propositional Logic (or) Boolean logic
  - ➢ Predicate logic (or) First order logic

2.1 Logical Agents:

- The Logical agent has to perform the following task using logic representation. The tasks are,
  - ✓ To know the current state of the world
  - ✓ How to infer the unseen properties of the world
  - ✓ New changes in the environment
  - ✓ Goal of the agent

✓ How to perform actions depends on circumstances

2.2 Propositional Logic:

- Each fact is represented by one symbol.
- Proposition symbols can be connected with Boolean connectives, to give more complex meaning. Connectives ,
  - **Λ** Logical Conjunction
  - **V** Logical disjunction
  - ¬ Negation
  - ⇔ Material Equivalence or Biconditional
  - ⇒ Material Implication or conditional
- Simple statements are implemented
- The Symbols of propositional logic are the logical constants, (True and False)
- For Example: - P, Q

Connectives

Λ (and)	**----- Example: - P Λ Q**
V (or)	**----- Example: - P V Q**
⇒ (implies)	**----- Example: - (P Λ Q) ⇒ R**
⇔ (equivalent)	**----- Example: - (P Λ Q) ⇔ (Q Λ P)**
¬ (not)	**----- Example: - ¬ P**

- A **BNF** (Backus-Naur Form) grammer of sentence in propositional logic

Sentence	**---------> Atomic sentence | complex sentence**
Atomic sentence	**---------> True | False | P | Q | R |**
Complex sentence	**---------> (sentence) | sentence connective sentence | ¬ Sentence**
Connective	**---------> Λ | V | ⇒ | ⇔**

- Order of precedence (from highest to lowest) : **¬ , Λ , V , ⇒ and ⇔**
- Example : - ¬P VQ ΛR ⇒ S is equivalent to ((¬P)V(Q Λ R)) ⇒ S
- The following truth table shows the five logical connectives

P	Q	¬ P	P Λ Q	P V Q	P ⇒ Q	P ⇔ 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

- These truth table can be used to define the validity of a sentence.
- If the sentence is true in every row (i.e. for different types of logical constants) then the sentence is a valid sentence.
- For Example: - ((P V H) Λ¬H) ⇒ P Check whether the given sentence is a valid sentence or not.

P	H	P V H	(P V H) Λ¬H	((P V H) Λ¬H) ⇒ P
False	False	False	False	True
False	True	True	False	True
True	False	True	True	True
True	True	True	False	True

- The given sentence is ((P V H) Λ¬H) ⇒ P valid sentence, because the sentence is TRUE in every row for different types of logical statements.

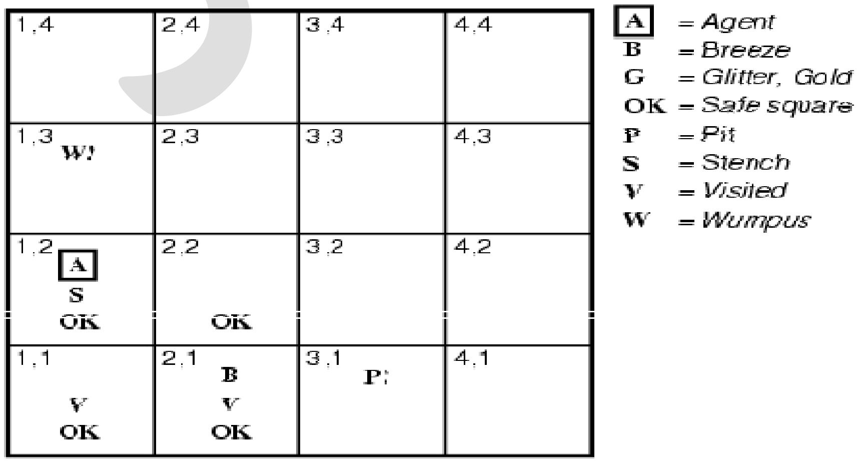
2.3 Inference Rules for Propositional logic:

- The propositional logic has seven inference rules.
- Inference means conclusion reached by reasoning from data or premises; speculation.
- A procedure which combines known facts to produce ("infer") new facts.
- **Logical inference** is used to create new sentences that logically follow from a given set of predicate calculus sentences (KB).
- An inference rule is **sound** if every sentence X produced by an inference rule operating on a KB logically follows from the KB. (That is, the inference rule does not create any contradictions)
- An inference rule is **complete** if it is able to produce every expression that logically follows from (is entailed by) the KB. (Note the analogy to complete search algorithms.)
- Here are some examples of sound rules of inference
  A rule is sound if its conclusion is true whenever the premise is true

Rule	Premise	Conclusion
Modus Ponens	$A, A \rightarrow B$	B
And Introduction	A, B	$A \wedge B$
And Elimination	$A \wedge B$	$A \vee B$
Or Introduction	A, B	A
Double Negation	$\neg\neg A$	A
Unit Resolution	$A \vee B, \neg B$	A
Resolution	$A \vee B, \neg B \vee C$	$A \vee C$

2.4 An Agent for the Wumpus world – Propositional logic

- We will discuss the knowledge base representation and a method to find the wumpus using propositional logic representation.
- From the following figure assume that the agent has reached the square (1,2)



- **The Knowledge Base:** The agent percepts are converted into sentences and entered into the knowledge base, with some valid sentences that are entailed by the percept sentences
- From the above figure we can perceive the following percept sentences and it is added to the knowledge base.

$\neg S_{1,1}$	$\neg B_{1,1}$	-----	for the square (1, 1)
$\neg S_{2,1}$	$\neg B_{2,1}$	-----	for the square (2, 1)
$S_{1,2}$	$\neg B_{1,2}$	-----	for the square (1, 2)
$S_{1,2}$		-----	There is a stench in (1, 2)
$\neg B_{1,2}$		-----	There is a breeze in (1, 2)
$\neg S_{2,1}$		-----	There is a stench in (2, 1)
$B_{2,1}$		-----	There is a breeze in (2, 1)
$\neg S_{1,1}$		-----	There is a stench in (1, 1)
$\neg B_{1,1}$		-----	There is a breeze in (1, 1)

- The rule of three squares,
  - R_{1} :- $\neg S_{1,1} \Rightarrow \neg W_{1\,1} \wedge \neg W_{1\,2} \wedge \neg W_{2\,1}$
  - R_2:- $\neg S_{2,1} \Rightarrow \neg W_{1\,1} \wedge \neg W_{2\,2} \wedge \neg W_{2\,1} \wedge \neg W_{3\,1}$
  - R_{3}:- $\neg S_{1,2} \Rightarrow \neg W_{1\,1} \wedge \neg W_{1\,2} \wedge \neg W_{2\,2} \wedge \neg W_{1\,3}$
  - R_{4}:- $\neg S_{1,2} \Rightarrow W_{1\,3} \vee W_{1\,2} \vee W_{2\,2} \vee W_{1\,1}$

- Finding the wumpus as, We can prove that the Wumpus is in (1, 3) using the four rules given.
  - Apply Modus Ponens with $\neg S_{1\,1}$ and R_1:
    - $\neg W_{1\,1} \wedge \neg W_{1\,2} \wedge \neg W_{2\,1}$
  - Apply And-Elimination to this, yielding three sentences:
    - $\neg W_{1\,1}, \neg W_{1\,2}, \neg W_{2\,1}$
  - Apply Modus Ponens to ¬S21 and R2, then apply And-elimination:
    - $\neg W_{2\,2}, \neg W_{2\,1}, \neg W_{3\,1}$
  - Apply Modus Ponens to S12 and R4 to obtain:
    - $W_{1\,3} \vee W_{1\,2} \vee W_{2\,2} \vee W_{1\,1}$
  - Apply Unit resolution on $(W_{1\,3} \vee W_{1\,2} \vee W_{2\,2} \vee W_{1\,1})$ and $\neg W_{1\,1}$:
    - $W_{1\,3} \vee W_{1\,2} \vee W_{2\,2}$
  - Apply Unit Resolution with $(W_{1\,3} \vee W_{1\,2} \vee W_{2\,2})$ and $\neg W_{2\,2}$:
    - $W_{1\,3} \vee W_{1\,2}$
  - Apply Unit Resolution with $(W_{1\,3} \vee W_{1\,2})$ and $\neg W_{1\,2}$:
    - $W_{1\,3}$

2.5 First-Order Logic:

- **First-Order Logic** is a logic which is sufficiently expressive to represent a good deal of our commonsense knowledge.

- It is also either includes or forms the foundation of many other representation languages.

- It is also called as **First-Order Predicate calculus.**

- It is abbreviated as **FOL** or **FOPC**

2.5.1 Representation Revisited:

- It is necessary to know about the nature of representation languages.
- The following are the some languages,
  - ✓ Programming languages
  - ✓ Propositional logic languages
  - ✓ Natural languages

Programming languages:

- Programming languages like C++ or Java are the largest class of formal languages in common use.
- Programs represent only computational processes.
- Data structures within programs can represent facts.
- For Example, 4 x 4 arrays can be used by a program to represent the contents of the Wumpus world.
- Thus the programming language statement *World [2, 2]* $\longleftarrow$ *Pit* is a fairly natural way to assert that there is a pit in square [2, 2].

Disadvantages:

- Programming languages lack is any general mechanism for deriving facts from other facts; each update to a data structure is done by a domain-specific procedure whose details are derived by the programmer from his or her own knowledge of the domain.
- A second drawback of data structures in programs is the lack of any easy way to say
- For Example, “There is a Pit in [2,2] or [3,1]” or “If the Wumpus is in [I,I] then he is not in [2,2]”.
- Programs lack the expressiveness required to handle partial information.

Propositional Logic Languages:

- Propositional logic is a declarative language
- The following are the properties of propositional logic
- Its semantics is based on a truth relation between sentences and possible worlds.
- It also has sufficient expressive power to deal with partial information, using disjunction and negation.

- It also has **compositionality** that is desirable in representation languages namely **compositionality.**
- In a **compositionality** language, the meaning of a sentence is function of the meaning of its parts.
- For Example, "$S_{1,4}$ **Λ** $S_{1,2}$" is related to the meanings of "$S_{1,4}$" and "$S_{1,2}$"
- It would be very strange if "$S_{1,4}$" meant that there is a stench in square [1,4] and "$S_{1,2}$" meant that there is a stench in square [1,2], but "$S_{1,4}$ **Λ** $S_{1,2}$" meant that France and Poland drew 1-1 in last week's ice hockey qualifying match.
- Clearly, non Compositionality makes life much more difficult for the reasoning system.

Advantages:

- Declarative
- Context-Independent
- Unambiguous

Disadvantages:

- It lacks the expressive power to describe an environment with many objects concisely.
- For Example, it is forced to write a separate rule about breezes and pits for each square, such as $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$.
- The procedural approach of programming languages can be contrasted with the declarative nature of propositional logic, in which knowledge and inference are separate and inference is entirely domain-independent.

Natural Languages:

- A moments thought suggests that natural languages like English are very expressive indeed.
- Natural language is essentially a declarative knowledge representation language and attempts to pin down its formal semantics.
- The modern view of natural language is that it serves a somewhat different purpose, namely as a medium for communication rather than pure representation.
- When a speaker points and says, "Look!" the listener comes to know that, say, Superman has finally appeared over the rooftops.
- The meaning of the above sentence depends both on the sentence itself and on the context in which the sentence was spoken.

Disadvantages:

- It is difficult to understand how the context can be represented.
- This is because one could not store a sentence such as "Look!" in knowledge base and expect to recover its meaning without also storing a representation of the context.
- They are also non-compositional
- They suffer from ambiguity, which would cause difficulties for thinking.
- For Example, when people think about spring, they are not confused as to whether they are thinking about a season or something that goes *boing-and* if one word can correspond to two thoughts, thoughts can't be words.

2.5.2 First-Order Logic:

- **First-Order Logic** is a logic which is sufficiently expressive to represent a good deal of our commonsense knowledge.
- It is also either includes or forms the foundation of many other representation languages.
- It is also called as **First-Order Predicate calculus.**
- It is abbreviated as **FOL** or **FOPC**
- **FOL** adopts the foundation of propositional logic with all its advantages to build a more expressive logic on that foundation, borrowing representational ideas from natural language while avoiding its drawbacks
- The Syntax of natural language contains elements such as,

  Nouns and noun phrases that refer to objects (Squares, pits, rumpuses)

  Verbs and verb phrases that refer to among objects (is breezy, is adjacent to)
- Some of these relations are functions-relations in which there is only one "Value" for a given "input".
- For Example,

  Objects: People, houses, numbers

  Relations: These can be unary relations or properties such as red, round,

  More generally n-ary relations such as brother of, bigger than,

  Functions: father of, best friend,…
- Indeed, almost any assertion can be thought of as referring to objects and properties or relations
- For Example, in the way of Sentence " One plus Two is Three"
- Where,

Objects: One, Two, Three, One plus Two

Relations: equals

Function: plus

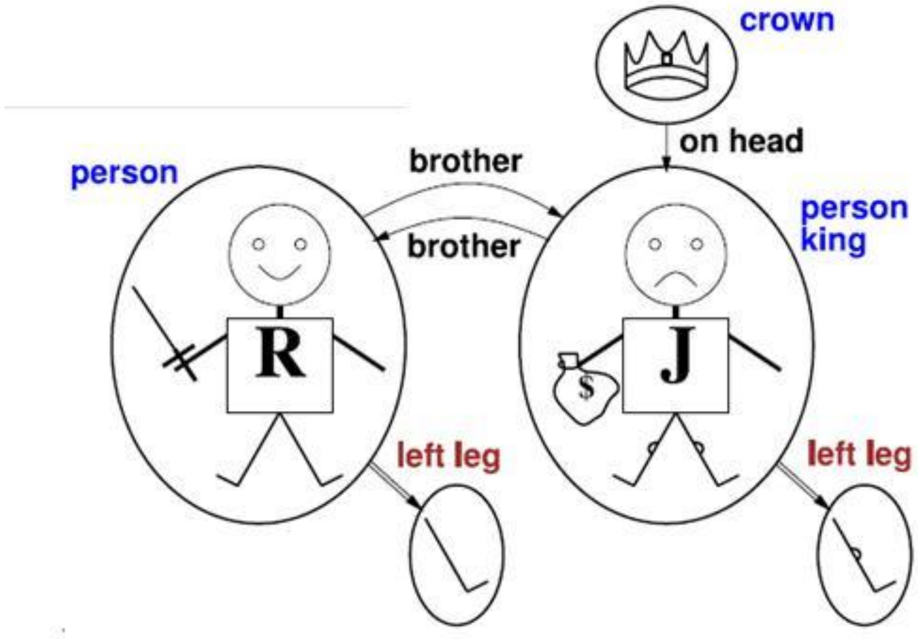
- Ontological commitment of First-Order logic language is "Facts", "Objects", and "Relations".
- Where ontological commitment means "WHAT EXISTS IN THE WORLD".
- Epistemological Commitment of First-Order logic language is "True", "False", and "Unknown".
- Where epistemological commitment means "WHAT AN AGENT BELIEVES ABOUT FACTS".

Advantages:

- It has been so important to mathematics, philosophy, and Artificial Intelligence precisely because those fields can be usefully thought of as dealing with objects and the relations among them.
- It can also express facts about some or all of the objects in the universe.
- It enables one to represent general laws or rules, such as the statement "Squares neighboring the wumpus are smelly".

2.5.3 Syntax and Semantics of First-Order Logic:

- The models of a logical language are the formal structures that constitute the possible worlds under consideration.
- Models for propositional logic are just sets of truth values for the proposition symbols.
- Models for first-order logic are more interesting.
- First they have objects in them.
- The domain of a model is the set of objects it contains; these objects are sometimes called domain elements.
- The following diagram shows a model with five objects



- The five objects are,
  - ✓ Richard the Lionheart
  - ✓ His younger brother
  - ✓ The evil King John
  - ✓ The left legs of Richard and John
  - ✓ A crown
- The objects in the model may be related in various ways, In the figure Richard and John are brothers.
- Formally speaking, a relation is just the set of tuples of objects that are related.
- A tuple is a collection of Objects arranged in a fixed order and is written with angle brackets surrounding the objects.
- Thus, the brotherhood relation in this model is the set

  {(Richard the Lionheart, King John),(King John, Richard the Lionheart)}
- The crown is on King John's head, so the "on head" relation contains just one tuple, (the crown, King John).
- The relation can be binary relation relating pairs of objects (Ex:- "Brother") or unary relation representing a common object (Ex:- "Person" representing both Richard and John)

- Certain kinds of relationships are best considered as functions that relates an object to exactly one object.
- For Example:- each person has one left leg, so the model has a unary "left leg" function that includes the following mappings

  (Richard the Lionheart) ----> Richard's left leg

  (King John) ----> John's left leg

➢ **Symbols and Interpretations:**

- The basic syntactic elements of first-order logic are the symbols that stand for **objects, relations** and **functions**

❖ **Kinds of Symbols**

- The symbols come in three kinds namely,
  - ✓ Constant Symbols standing for **Objects** (Ex:- Richard)
  - ✓ Predicate Symbols standing for **Relations** (Ex:- King)
  - ✓ Function Symbols stands for **functions** (Ex:- LeftLeg)
    - o Symbols will begin with uppercase letters
    - o The choice of names is entirely up to the user
    - o Each predicate and function symbol comes with an arity
    - o Arity fixes the number of arguments.
- The semantics must relate sentences to models in order to determine truth.
- To do this, an interpretation is needed specifying exactly which **objects, relations** and **functions** are referred to by the **constant, predicate and function symbols**.
- One possible interpretation called as the intended interpretation- is as follows;
  - ✓ **Richard** refers to **Richard the Lionheart** and **John** refers to the **evil King John.**
  - ✓ **Brother** refers to the brotherhood relation, that is the set of tuples of objects given in equation **{(Richard the Lionheart, King John),(King John, Richard the Lionheart)}**
  - ✓ **OnHead** refers to the "on head" relation that holds between the crown and King John; **Person, King** and **Crown** refer to the set of objects that are persons, kings and crowns.
  - ✓ **Leftleg** refers to the "left leg" function, that is, the mapping given in **{(Richard the Lionheart, King John),(King John, Richard the Lionheart)}**

- There are many other possible interpretations relating these symbols to this particular model.
- The truth of any sentence is determined by a model and an interpretation for the sentence symbols.
- The syntax of the FOL with equality specified in BNF is as follows

Sentence	$\longrightarrow$	AtomicSentence
	|	(Sentence Connective Sentence)
	|	Quantifier Variable,…Sentence
	|	**-** Sentence
AtomicSentence	$\longrightarrow$	Predicate (Term…) | Term = Term
Term	$\longrightarrow$	Function (Term,….)
	|	Constatnt
	|	Variable
Connective	$\longrightarrow$	**⇒ | Λ | V | ⇔**
Quantifier	$\longrightarrow$	**∀ | ∃**
Constant	$\longrightarrow$	**A | X_1 | John**
Variable	$\longrightarrow$	**a | x | s | …**
Predicate	$\longrightarrow$	**Before | HasColor | Raining | ….**
Function	$\longrightarrow$	**Mother | LeftLeg | ….**

- Where,

Λ	Logical Conjunction
V	Logical disjunction
∀	Universal Quantification
∃	Existential Quantification
⇔	Material Equivalence
⇒	Material Implication

❖ **Terms:**

- A Term is a logical expression that refers to an object
- Constant symbol are therefore terms, but it is not always convenient to have a distinct symbol to name every object.
- For Example:- in English we might use the expression “King Johns left leg” rather than giving a name to his leg.

- A complex term is formed by a function symbol followed by a parenthesized list of terms as arguments to the function symbol.
- It is just like a complicated kind of name. It's not a "subroutine call" that returns a value".
- The formal semantics of terms is straight forward,

Consider a term $f(t1……t_n)$

Where

f- some function in the model (call it F)

The argument terms – objects in the domain

The term – object that is the value of the function F applied to the domain

- For Example:- suppose the **LeftLeg** function symbol refers to the function is,

(Richard the Lionheart) ------ Richards left leg

(King John) ------ Johns left leg

John refers to King John, then **LeftLeg** (John) refers to king Johns left leg.

- In this way the Interpretation fixes the referent of every term.

❖ **Atomic Sentences:-**

- An atomic sentence is formed from a predicate symbol followed by a parenthesized list of terms:

Brother (Richard, John)

- This states that Richard the Lionheart is the brother of King John.
- Atomic Sentences can have complex terms as arguments.
- Thus, **Married(Father(Richard), Mother(John))** states that Richard the Lionheart's father is married to King John's mother
- An atomic sentence is true in a given model, under a given interpretation, if the relation referred to by the predicate symbol holds among the objects referred to by the arguments.

❖ **Complex Sentences:-**

- Logical connectives can be used to construct more complex sentences, just as in propositional calculus.
- The semantics of sentences formed with logical connectives is identical to that in the propositional case.

¬ Brother (LeftLeg (Richard), John)

Brother (Richard, John) Λ Brother (John, Richard)

King (Richard) V King (John)

¬ King (Richard) ⇒ King (John)

¬ **it refers " Logical Negation"**

❖ **Quantifiers:-**

- Quantifiers are used to express properties of entire collections of objects, instead of enumerating the objects by name if a logic that allows object is found.
- It has two type,
- The following are the types of standard quantifiers,
    - ✓ Universal
    - ✓ Existential

❖ **Universal Quantification (∀):-**

- Universal Quantification make statement about every object.
- "All Kings are persons", is written in first-order logic as

$\forall_x$ king (x) ⇒ Person (x)

- ∀ is usually pronounced "For all….", Thus the sentences says , "For all x, if x is a king, then x is a person".
- The symbol x is called a variable.
- A variable is a term all by itself, and as such can also serve as the argument of a function-for example, **LeftLeg(x)**.
- A term with no variables is called a **ground term.**
- Based on our model, we can extend the interpretation in five ways,

x --------- Richard the Lionheart

x --------- King John

x --------- Richard's Left leg

x --------- John's Left leg

x --------- the crown

- The universally quantified sentence is equivalent to asserting the following five sentences

Richard the Lionheart ------ Richard the Lionheart is a person

King John is a King ------ King John is a Person

Richard’s left leg is King -------- Richard’s left leg is a person

John’s left leg is a King -------- John’s left leg is a person

The crown is a King -------- The crown is a Person

❖ **Existential Quantification (∃):-**

- An existential quantifier is used make a statement about some object in the universe without naming it.
- To say, for example :- that King John has a crown on his head, write ∃x crown (x) Λ OnHead (x, John).
- ∃x is pronounced “ There exists an x such that..,” or “For some x..”
- Consider the following sentence,

$\exists_x$ crown (x) ⇒ OnHead (x,John)

- Applying the semantics says that at least one of the following assertion is true,

Richard the Lionheart is a crown	Λ Richard the Lionheart is on John’s head
King John is Crown	Λ King John is on John’s head
Richard’s left leg is a crown	Λ Richard’s left leg is on John’s head
John’s left leg is a crown	Λ John’s left leg is on John’s head
The crown is a crown	Λ The crown is on John’s head

- Now an implication is true if both premise and conclusion are true, or if its premise is false.

❖ **Nested Quantifiers:-**

- More complex sentences are expressed using multiple quantifiers.
- The following are the some cases of multiple quantifiers,
- The simplest case where the quantifiers are of the same type.
- For Example:- “Brothers are Siblings” can be written as

$\forall_x \forall_{y,}$ Brother (x,y) ⇒ sibling (x,y)

- Consecutive quantifiers of the same type can be written as one quantifier with several variables.
- For Example:- to say that siblinghood is a symmetric relationship as

$\forall_{x,y}$ sibling (x,y) ⇔ sibling (y,z)

- In some cases it is possible to have mixture of quantifiers.
- For Example:- “Everybody loves somebody” means that for every person, there is someone that person loves:

$\forall_x \exists_y$ Loves (x, y).

- On the other hand , to say "There is someone who is loved by everyone", we can write as

$\exists_y \forall_x$ Loves (x, y).

❖ **Connections between ∀ and ∃**

- The two quantifiers are actually intimately connected with each other , through negation,
- Declaring that everyone dislikes parsnips is the same as declaring there does not exist someone who likes them, and vice versa:

$\forall_x \neg$ Likes (x, Parsnips) is equivalent to $\neg \exists_x$ Likes (x, Parsnips)

- Going one step further: "Everyone likes ice cream" means that there os no one who does not like ice cream:

$\forall_x \neg$ Likes (x, IceCream) is equivalent to $\neg \exists_x$ Likes (x, IceCream)

- Because ∀ is really a conjunction over the universe of objects ∃ is a disjunction, it should not be surprising that they obey de Morgan's rules.
- The de Morgan's rules for quantified and un-quantified sentences are as follows:
- **≡ it refers definition**

$\forall_x \neg P \equiv \neg \exists_x P$	$\neg P \wedge \neg Q \equiv \neg (P \vee Q)$
$\neg \forall_x P \equiv \exists_x \neg P$	$\neg (P \wedge Q) \equiv \neg P \vee \neg Q$
$\forall_x P \equiv \neg \exists_x \neg P$	$P \wedge Q \equiv \neg (\neg P \vee \neg Q)$
$\exists_x P \equiv \neg \forall_x \neg P$	$P \vee Q \equiv \neg (\neg P \wedge \neg Q)$

❖ **Equality:-**

- First – order logic includes one more way of using equality symbol to make atomic sentences.
- Use of equality symbol
  - ✓ The equality symbol can be used to make statements to the effect that two terms refer to the same object.
  - ✓ For Example: - Father (John) = Henry says that the object referred to by Father (John) and the object referred to by Henry are the same.
  - ✓ Determining the truth of an equality sentence is simply a matter of seeing that the referents of the two terms are the same object.
  - ✓ The equality symbol can be used to state facts about a given function

- ✓ It can also be used with negation to insist that two terms are not the same object.
- ✓ For Example:- To say that Richard has at least two brothers, write as

$\exists_{x,y}$ Brother(x, Richard) Λ Brother(y,Richard) $\Lambda \neg (x = y)$

2.5.4 Using First – Order Logic

The best way to learn about FOL is through examples. In Knowledge representation, a domain is just some part of the world about which some knowledge is to be expressed.

❖ **Assertions:-**

- Sentences that are added to knowledge base using **TELL**, exactly as in propositional logic are called assertion (Declaration/Statement).
- For Example:- It can be declared that " John is a King and that Kings are persons"

TELL (KB, King(John))
TELL (KB, $\forall_x$ King(x) $\Rightarrow$ Person(x))

❖ **Queries:-**

- Questions of the knowledge base can be asked using **ASK.**
- For Example:- **ASK(KB, King(John))** returns **true**.
- Questions asked using **ASK** are called queries or goals.
- Generally speaking, any query that is logically needed by the knowledge base should be answered positively.
- For Example:- Given the two assertions in the preceding line, the query

ASK(KB, Person(John) should also return **true**

❖ **Substitution/Binding List:-**

- Substitution or Binding list is a set of variable/term pairs.
- It is a standard form for an answer of a query with existential variables.
- For Example:- "Is there an x such that…" is solved by providing such an x.
- Given Just the two assertions, the answer would be {x/John}
- If there is more than one possible answer, a list of substitutions can be returned.

❖ **The Kinship Domain:-**

- Kinship domain is the domain of family relationships, or Kinship.
- This domain includes facts such as "Elizabeth is the mother of Charles" and "Charles is the father of William" and rules such as "One's grandmother is the mother of one's parent".
- The objects in this domain are people.
- There will be two unary predicates as **"Male"** and **"Female"**
- Kinship relations will be represented by binary predicates.

- For Example:- parenthood, brotherhood, marriage and so on are represented by Parent, sibling, Brother, Sister, Child, Daughter,Son,Spouse,Wife, Husband, Grandparents, Grandchild, Cousin, Aunt, and Uncle.
- For Example:-
- **One's mother is One's female parent:**

$$\forall m, c\ Mother(c) = m \Leftrightarrow Female\ (m) \land Parent\ (m,c)$$

❖ **Axioms:-**

- Axioms are commonly associated with purely mathematical domains.
- The axioms define, the Mother function and the Husband, Male, Parent and Sibling predicates in terms of other predicates.
- They provide the basic factual information from which useful conclusions can be derived.
- Kinship axioms are also definitions : they have the form $\forall_{x,y}$ **p(x,y)** ⇔ ...

❖ **Theorems:-**

- Not all logical sentences about a domain are axioms.
- Some are Theorems-that is, they are caused by the axioms.
- For Example:-

$$\forall_{x,y}\ Sibling(x,y) \Leftrightarrow \ldots Sibling\ (y,x)$$

- The above declaration that siblinghood is symmetric
- It's a theorem that follows logically from the axiom that defines siblinghood.
- If ASK Questions the knowledge base this sentence, it should return true
- From logical point of view, a knowledge base need contain only axioms and no theorems
- From a practical point of view, theorems are essential to reduce the computational cost of deriving new sentences.

❖ **Numbers:-**

- Numbers are perhaps the most brilliant example of how a large theory can be built up from a tiny heart of axioms.
- Requirements
    - ✓ A predicate ***NatNum*** is needed that will be true of natural numbers
    - ✓ One constant symbol, 0
    - ✓ One function symbol, S (Successor)

- **Peano Axioms:-**
    - ✓ The peano axioms define natural numbers and addition.
    - ✓ Natural numbers are defined recursively:

$$NatNum\ (0)$$

$$\forall_n \text{ NatNum}(n) \Rightarrow \text{NatNum}(S(n))$$

- ✓ That is, 0 is a natural number, and for every object n, if n is a natural number then S(n) is a natural number,
- ✓ So the natural numbers are 0, S(0), S(S(0)), and so on..

❖ **Sets:-**

- The domain of sets is also fundamental to mathematics as well as to commonsense reasoning
- The empty set is a constant written as { }.
- There is one unary predicate, Set, which is true of sets.
- The binary predicates are x € s (**x** is a member of set **s**) and s1 ⊆ s2 (set s1 is a subset, not necessarily proper, of set s2)
- The binary functions are s1 ∩ s2 (the intersection of two sets), s1 ∪ s2 (the union of two sets), and {x/s} (the set resulting from adjoining element x to set **s**)
- One possible set of axioms is as follows,

  - ✓ The only sets are the empty set and those made by adjoining something to a set

    $$\forall_s \textbf{ Set(s)} \Leftrightarrow (s = \{\,\})\ \textbf{V}\ (\exists_x, s2\ \textbf{Set(s2)}\ \Lambda\ s = \{x/s2\}$$

  - ✓ There is no way to decompose EmptySet into a smaller set and an element:

    $$\neg \exists_{x,}\ s\ \{x/s\} = \{\,\}$$

  - ✓ Adjoining an element already in the set has no effect:

    $$\forall_{x,s}\quad x \in \text{(set membership) } s \Leftrightarrow s = \{x/s\}$$

  - ✓ The only members of a set are the elements that were connected into it. This can be expressed recursively, saying that **x** is a member of **s** if and only if **s** is equal to some set S2 connected with some element **y**, where either **y** is the same as **x** or **x** is a member of S2

    $$\forall_{x,s}\quad x \in s \Leftrightarrow [\exists_{y,}\ s2\ (s=\{y/s2\}\ \Lambda\ (x=y\ \textbf{V}\ x \in s2))\ ]$$

  - ✓ A set is subset of another set if and only if all of the first sets members are members of the second set

    $$\forall_{s1,\,s2}\quad s1 \subseteq s2 \Leftrightarrow (\forall_X\ x \in s1 \Rightarrow x \in s2)$$

  - ✓ Two sets are equal if and only if each is a subset of the other

    $$\forall_{s1,\,s2}\quad (s1 = s2) \Leftrightarrow (s1 \subseteq s2\ \Lambda\ s2 \subseteq s1)$$

- ✓ An object is in the Intersection of two sets if and only if it is a member of both sets

$$\forall_{x,\ s1,\ s2}\quad x \in (s1 \cap s2)\qquad (x \in s1\ \Lambda\ x \in s2)$$

- ✓ An object is in the union of two sets if and only if it is a member of either set

$$\forall_{x,\ s1,\ s2}\quad x \in (s1 \cup s2)\qquad (x \in s1\ V\ x \in s2)$$

❖ **Lists:-**

- Lists are similar to sets.
- The differences are that lists are ordered and the same element can appear more than once in a list.
- **Nil** is the constant list with no elements
- **Cons, Append, First,** and **Rest** are functions.
- **Find** is the predicate that does for lists what ***Member*** does for sets.
- **List?** is a predicate that is true only of lists.
- The empty list is **f1.**
- The term **Cons (x, y),** where y is a nonempty list, is written **[x/y]**.
- The term **Cons (x, Nil)**, (i.e. The list containing the element **x**), is written as **x1.**
- A list of several elements, such as [A,B,C] corresponds to the nested term **Cons(A, Cons(B, Cons(C, Nil)))**

❖ **The Wumpus World:-**

- The first order axioms of wumpus world are more concise, capturing in a natural way what exactly we want to represent the concept.
- Here the more interesting question is **"how an agent should organize what it knows in order to take the right actions".**
- For this purpose we will consider three agent architectures:

✓ Reflex agents	- classify their percept and act accordingly
✓ Model based agents	- construct an internal representation of the World and use it to act
✓ Goal based agents	- form goals and try to achieve them

- The first step of wumpus world agent construction is to define the interface between the environment and the agent.
- The percept sentence must include both the percept and the time at which it occurred, to differentiate between the consequent percepts.
- For Example:-

Percept ([Stench, Breeze, Glitter, None, None], 3)

- In this sentence

Percept	-	**predicate**

Stench, Breeze, Glitter	**-**	**Constants**
3	**-**	**Integer to represent to time.**

- The agents action are,
  Turn (Right)
  Turn (Left)
  Forward
  Shoot
  Grab
  Release
  Climb
- To determine which is best, the agent program constructs a query such as
  $\exists_a$ **BestActions(a,5)**
- **ASK** solves this query and returns a binding list such as {a/Grab}.
- The agent program then calls **TELL** to record the action which was taken to update the Knowledge base **KB.**

❖ **Types of Sentences:-**

- The percept sentences are classified in to two as,
  - ✓ **Synchronic (Same time)**
  - ✓ **Diachronic (across time)**
- **Synchronic: -** The sentences dealing with time is synchronic if they relate properties of a world state to other properties of the same world state.
- **Diachronic: -** The sentences describing the way in which the world changes (or does not change) are diachronic sentences

❖ **Kinds of Synchronic Rules:-**

- There are two kinds of synchronic rules that could allow to capture the necessary information for deductions are,

  - ✓ Diagnostic rules
  - ✓ Casual rules

  - o **Diagnostic Rules: -** Infer the presence of hidden properties directly from the percept – derived (observed) information.
  - o For Example: **-** For finding pits, if a square is breezy, some adjacent square must contain a pit.

$$\forall_s \textbf{ Breezy(s)} \Rightarrow \exists_r \textbf{ Adjacent (r,s)} \land \textbf{ Pit(r)}$$

  - o If a square is not breezy, no adjacent square contains a pit.

$$\forall_s \neg\textbf{Breezy(s)} \Rightarrow \neg\exists_r \textbf{ Adjacent (r,s)} \land \textbf{ Pit(r)}$$

  - o Combining these two, the derived biconditional sentence is :

$$\forall_s \textbf{ Breezy(s)} \Leftrightarrow \exists_r \textbf{ Adjacent (r,s)} \land \textbf{ Pit(r)}$$

- **Casual Rules: -** Reflect the assumed direction of casuality in the world. Some hidden property of the world causes certain percepts to be generated.
- For Example:- A Pit causes all adjacent squares to be breezy.

$$\forall_r \, Pit(r) \Rightarrow [\forall_s \, Adjacent\ (r, s) \Rightarrow Breezy(s)]$$

- If all squares adjacent to a given square are pitless, the square will not be breezy

$$\forall_s \, [\forall_r \, Adjacent\ (r,s) \Rightarrow \neg\, Pit(r)] \Rightarrow \neg\, Breezy(s)$$

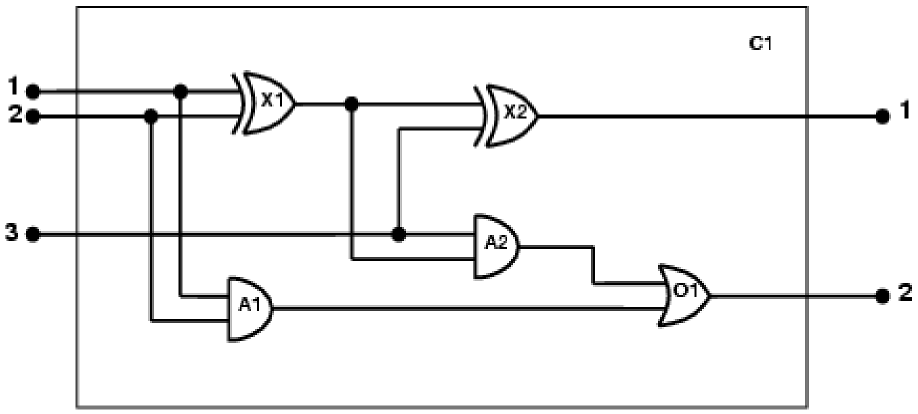
- System that reason with casual rules are called model-based reasoning systems, because the casual rules from a model of how the environment operates.

2.5.5 KNOWLEDGE ENGINEERING IN FIRST ORDER LOGIC:-

- **Knowledge Engineering: -** The general process of Knowledge Base **KB** Construction.
- **Knowledge Engineer: -** Who investigates a particular domain, learns what concepts are important in that domain and creates a formal representation of the objects and relations in the domain.
- The knowledge engineering projects vary widely in content, scope and difficulty, but all projects include the following steps,

  - ✓ **Identify the Task: -** The Knowledge engineer should identify the **PEAS** description of the domain.
  - ✓ **Assemble the relevant knowledge: -** The idea of combining expert's knowledge of that domain (i.e.) a process called **knowledge acquisition.**
  - ✓ **Decide on a vocabulary of predicates, functions and constants: -** Translate the important domain-level concepts into logical level name. The resulting vocabulary is known as **ontology** of the domain, which determines what kinds of things exist, but does not determine their specific properties and interrelationships.
  - ✓ **Encode general knowledge about the domain: -** The knowledge engineer writes the axioms (rules) for all the vocabulary terms. The misconceptions are clarified from step 3 and the process is iterated.
  - ✓ **Encode a description of the specific problem instance: -** To write simple atomic sentences about instances of concepts that are already part of the ontology.
  - ✓ **Pose queries to the inference procedure and get answers: -** For the given query the inference procedure operate on the axioms and problem specific facts to derive the answers.
  - ✓ **Debug the knowledge base: -** For the given query , if the result is not a user expected on then **KB is** updated with relevant or missing axioms.

- The seven step process is explained with the domain of **ELECTRONIC CIRCUITS DOMAIN.**



- ✓ **Identify the task: -**
  - o Analyse the circuit functionality, does the circuit actually add properly? (Circuit Verification).
- ✓ **Assemble the relevant knowledge: -**
  - o The circuit is composed of wire and gates.
  - o The four types of gates (AND, OR, NOT, XOR) with two input terminals and one output terminal knowledge is collected.
- ✓ **Decide on a vocabulary: -**
  - o The functions, predicates and constants of the domain are identified.
  - o Functions are used to refer the type of gate.

    Type (x1) = XOR,

    where x1 ---- Name of the gate and Type ---- function
  - o The same can be represented by either binary predicate (or) individual type predicate.

    Type (x1, XOR) – binary predicate

    XOR (x1) – Individual type
  - o A gate or circuit can have one or more terminals. For x1, the terminals are x1In1, x1In2 and x1 out1

    Where x1 In1 ----- 1st input of gate x1

    x1 In2 ----- 2nd input of gate x1

    x1 out1 ---- output of gate x1
  - o Then the connectivity between the gates represented by the predicate connected. (i.e.) connected (out(1, x1), In(1,x2)).
  - o Finally the possible values of the output terminal C1, as true or false, represented as a signal with 1 or 0.
- ✓ **Encode general knowledge of the domain:-**
  - o This example needs only seven simple rules to describe everything need to know about circuits
  - o If two terminals are connected, then they have the same signal:
    - ▪ $\forall t_1,t_2$ Connected(t_1, t_2) $\Rightarrow$ Signal(t_1) = Signal(t_2)
  - o The signal at every terminal is either 1 or 0 (but not both):
    - ▪ $\forall t$ Signal(t) = 1 $\vee$ Signal(t) = 0
    - ▪ $1 \neq 0$

- Connected is a commutative predicate
  - $\forall t_1,t_2$ Connected(t_1, t_2) $\Rightarrow$ Connected(t_2, t_1)
- An OR gate's output is 1 if and only if any of its input is 1:
  - $\forall$g Type(g) = OR $\Rightarrow$ Signal(Out(1,g)) = 1 $\Leftrightarrow$ $\exists$n Signal(In(n,g)) = 1
- An AND gate's output is 0 if and only if any of its inputs is 0
  - $\forall$g Type(g) = AND $\Rightarrow$ Signal(Out(1,g)) = 0 $\Leftrightarrow$ $\exists$n Signal(In(n,g)) = 0
- An XOR gate's output is 1 if and only if inputs are different:
  - $\forall$g Type(g) = XOR $\Rightarrow$ Signal(Out(1,g)) = 1 $\Leftrightarrow$ Signal(In(1,g)) $\neq$ Signal(In(2,g))
- A NOT gate's output is different from its input:
  - $\forall$g Type(g) = NOT $\Rightarrow$ Signal(Out(1,g)) $\neq$ Signal(In(1,g))

✓ **Encode the specific problem instance:**

- First, we categorize the gates:

Type(X_1) = XOR Type(X_2) = XOR

Type(A_1) = AND Type(A_2) = AND

Type(O_1) = OR

- Then, we show the connections between them

Connected(Out(1,X_1),In(1,X_2)) Connected(In(1,C_1),In(1,X_1))

Connected(Out(1,X_1),In(2,A_2)) Connected(In(1,C_1),In(1,A_1))

Connected(Out(1,A_2),In(1,O_1)) Connected(In(2,C_1),In(2,X_1))

Connected(Out(1,A_1),In(2,O_1)) Connected(In(2,C_1),In(2,A_1))

Connected(Out(1,X_2),Out(1,C_1)) Connected(In(3,C_1),In(2,X_2))

Connected(Out(1,O_1),Out(2,C_1)) Connected(In(3,C_1),In(1,A_2))

✓ **Pose Queries to the inference procedure:**

- What combinations of inputs would cause the first output of C1(the sum bit) to be 0 and the second output of C1 (the carry bit) to be 1?

  $\exists i_1,i_2,i_3$ Signal(In(1,C1)) = $i_1 \wedge$ Signal(In(2,C_1)) = $i_2 \wedge$ Signal(In(3,C_1)) = $i_3 \wedge$ Signal(Out(1,C_1)) = o $\wedge$ Signal(Out(2,C_1)) = 1

- The answers are substitutions for the variables i_1,i_2 and i_3 Such that the resulting sentence is entailed by the knowledge base.
- There are three such substitutions as:

  { $i_{1/}1$,$i_{2/}1$, $i_3/0$} { $i_{1/}1$,$i_{2/}0$, $i_3/1$} { $i_{1/}0$,$i_{2/}1$, $i_3/1$}

- What are the possible sets of values of all the terminals for the adder circuit?

  $\exists i_1,i_2,i_3,o_1,o_2$ Signal(In(1,C1)) = $i_1 \wedge$ Signal(In(2,C_1)) = $i_2 \wedge$ Signal(In(3,C_1)) = $i_3 \wedge$ Signal(Out(1,C_1)) = $o_1 \wedge$ Signal(Out(2,C_1)) = o_2

✓ **Debug the knowledge base:**

- The knowledge base is checked with different constraints.
- For Example:- if the assertion $1 \neq 0$ is not included in the knowledge base then it is variable to prove any output for the circuit, except for the input cases 000 and 110.

2.6 Inference in First-order Logic:-

- We have learned seven inference rules of propositional logic.
- These rules are applicable for First-order logic also
- With those rules First-order logic holds some additional rules "with quantifiers" in which substituting particular individual for the variable is done (i.e.) SUBST(Θ, α) to denote the result of applying the substitution (or) binding list Θ to the sentence α.
- For Example:-

  SUBST ({x/Ram, y/John} Likes(x, y)) = Likes(Ram, John)
- The following are the new three inference rules for First-order Logic.
  - ✓ Universal Elimination
  - ✓ Existential Elimination
  - ✓ Existential Introduction
- **Universal Elimination:-** For any sentence α, variable v and ground term g;

  ∀v α / SUBST({v/g}, α)

  Example:-

  ∀x likes (x, Icecream) is a sentence **α**.

SUBST ({x/John}, α) is a substitution Ө = John
Likes (John, Icecream) – Inferred sentence

- **Existential Elimination :-** For any sentence α variable v, and constant symbol k that does not appear elsewhere in the Knowledge base:

  ∃v α / SUBST({v/k}, α)
  Example:-
  ∃x Kill (x, Victim) – **α**
  SUBST({x/Murderer, Victim}, **α**) where **Ө** = Murderer
  Kill (Murderer, Victim) – Inferred Sentence

- **Existential Introduction :-** For any sentence α, variable v that does not occur in **α**, and ground term **g** that does occur in **α**

  α / ∃v SUBST({g/v}, α)
  Example:-
  Likes (John, Icecream) – **α**
  ∃x Likes (x, Icecream) – Inferred Sentence

2.6.1 AN EXAMPLE PROOF USING FIRST- ORDER LOGIC:-

- An application of inference rule is matching their premise patterns to the sentences in the KB and then adding their conclusion patterns to the KB.
- **Task: -** For the given situation described in English, Convert it into its equivalent FOL representation and prove that "West is a Criminal".
- **Situation: -** The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.
- **Solution: -** The given description is splitted into sentences and is converted into its corresponding FOL representation.

  - ✓ It is a crime for an American to sell weapons to hostile nations:
    ∀x y z American(x) ∧ Weapon(y) ∧ Nation(z) ∧ Hostile(z) ∧ Sells(x, y, z ⇒ Criminal(x)
  - ✓ Nono … has some missiles,
    ∃x Owns(Nono,x) ∧ Missile(x)
  - ✓ all of its missiles were sold to it by Colonel West
    ∀x Missiles(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)
  - ✓ We will also need to know that missiles are weapons
    ∀x Missile(x) ⇒ Weapon(x)
  - ✓ An enemy of America counts as "hostile"
    ∀x Enemy(x,America) ⇒ Hostile(x)
  - ✓ West, who is American …
    American(West)

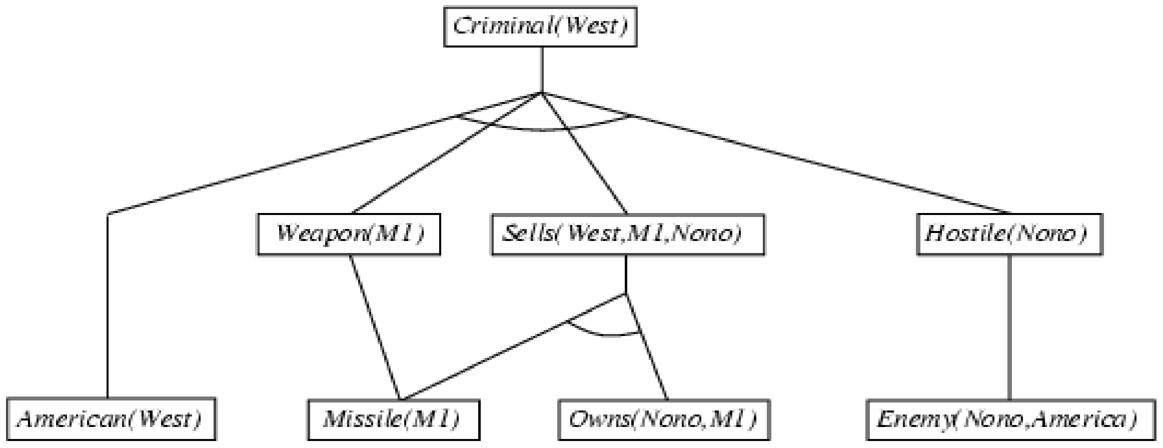
- ✓ Nono, is a nation
  Nation (Nono)
- ✓ Nono, an enemy of America
  Enemy (Nono, America)
- ✓ America is nation
  Nation (America)

2.7 Forward Chaining:-

- The Generalized Modus Ponens rule can be used by Forward Chaining.
- From the sentences in the **KB** which in turn derive new conclusions.
- Forward chaining is preferred when new fact is added to the database and we want to generate its consequences.
- **Forward Chaining Algorithm:-**
  - ✓ Forward chaining is triggered by the addition of new fact "p" into the knowledge base (i.e.) the action **TELL** is performed.
  - ✓ If the new fact is a rename of any other existing sentence in the **KB** then it is not included in **KB.**
  - ✓ With the new fact "p" find all premises that had "p" as premise and if any other premise is already known to hold then its consequence is included in **KB.**
  - ✓ The important operation of forward chaining is renaming : One sentence is a renaming of another if, they are identical except for the names of the variables.
  - ✓ **For Examples:-**
    - o Likes(x, Icecream) and Likes(y, Icecream) are renaming of each other.
    - o Likes(x,x) and Likes(x,y) are not renaming of each other (i.e.) its variable differs, the meaning is logically different.
  - ✓ Consider the **KB** of crime problem represented in Horn form to explain the concept of forward chaining.
  - ✓ The implication sentences are (i), (iv), (v), (vi)
  - ✓ Two iterations are required:
  - ✓ On the first iteration,
    - o Step (i) has unsatisfied premises
    - o Step (iv) is satisfied with {x/M1} and sells (west, M1,Nono) is added
    - o Step (v) is satisfied with {x/m1} and weapon (M1) is added
    - o Step (vi) is satisfied with {x/Nono}, and Hostile (Nono) is added
  - ✓ On the second iteration,
    - o Step (i) is satisfied with {x/West, y/M1, z/Nono} and Criminal(west) is added.
  - ✓ The following table shows the forward chaining algorithm,
  - ✓ **Inputs:- KB**, the Knowledge base, a set of first-order definite clauses **α,** the query, an atomic sentence
  - ✓ **Local variable:- new,** the new sentences inferred on each iteration

```
function FOL-FC-ASK($KB, \alpha$) returns a substitution or $false$
    repeat until $new$ is empty
        $new \leftarrow \{\}$
        for each sentence $r$ in $KB$ do
            $(p_1 \land \ldots \land p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART($r$)
            for each $\theta$ such that $(p_1 \land \ldots \land p_n)\theta = (p'_1 \land \ldots \land p'_n)\theta$
                        for some $p'_1, \ldots, p'_n$ in $KB$
                $q' \leftarrow$ SUBST($\theta, q$)
                if $q'$ is not a renaming of a sentence already in $KB$ or $new$ then do
                    add $q'$ to $new$
                    $\phi \leftarrow$ UNIFY($q', \alpha$)
                    if $\phi$ is not $fail$ then return $\phi$
        add $new$ to $KB$
    return $false$
```

- ✓ The following figure shows the proof tree generated by forward chaining algorithm on the Crime Example,



- ✓ The above discussed inference processes are not directed towards solving any particular problem: for this reason it is called a **data-driven or data-directed procedure.**
- ✓ In this problem, no new inferences are possible because every sentence concluded by forward chaining is already exist in the **KB.** Such a **KB** is called a **fixed point** of the inference process.
- ✓ FOL-FC-ASK function is sound and complete.
- ✓ Every inference is an application of generalized modus ponens, which is sound.
- ✓ Then it is complete for definite clauses **KB** (i.e.) it answers every query whose answers are entailed by any **KB** of definite clauses.

2.7.1 Efficient Forward chaining:-

- The above discussed FC Algorithm has three possible types of complexity.

- ✓ **Pattern Matching: -** "inner loop" of the algorithm involves finding all possible unifiers such that the premise of a rule unifies with a suitable set of facts in the **KB.**
- ✓ **Matching rules against known facts:-** The algorithm re-checks every rule on every iteration to see whether its premises are satisfied, even if very few additions are made to the **KB** on each iteration
- ✓ Irrelevant facts to the goal are generated

- In forward chaining approach, inference rules are applied to the knowledge base, yielding new assertions.
- This process repeats forever or until some stopping criterian is met.
- This method is appropriate for the design of an agent, that is on each cycle, we add the percepts to the knowledge base and run the forward chainer, which chooses an action to perform according to a set of condition action rules.
- Theoretically a **production system** can be implemented with a theorem prover, using resolution to do forward chaining over a full first order knowledge base.
- An efficient language can be used to perform this task, because it reduces the branching factor.
- The typical production system has three features:
  - ✓ The system maintained a **KB** called the working memory which has a set of positive literals with no variable
  - ✓ The system maintains a **rule memory**. This contains a set of inference rules $P_1 \wedge P_2 \Rightarrow act_1 \wedge act_2 \ldots$. That act_i is executed when p_i is satisfied, which performs adding or deleting an element from the **working memory – match phase.**
  - ✓ In each cycle, the system computes the subset of rules whose left-hand side is satisfied by the current contents of the **working memory - match phase.**

2.8 Backward Chaining:-

- Backward chaining is designed to find all answers to a question asked to the knowledge base. Therefore it requires a **ASK** procedure to derive the answer.
- The procedure **BACK WARD-CHAIN** will check two constraints.
  - ✓ If the given question can derive a answer directly from the sentences of the knowledge base then it returns with answers.
  - ✓ If the first constraint is not satisfied then it finds all implications whose conclusion unifies with the query and tries to establish the premises of those implications. If the premise is a conjunction then BACK-CHAIN processes the conjunction conjunct by conjunct, building up the unifiers for the whole premises as it goes.
- **Composition of Substitutions:-** COMPOSE(Θ_1, Θ_2) is the substitution whose effect is identical to the effect of applying each substitution in turn (i.e.),

  SUBST (COMPOSE (Θ_1, Θ_2), p) = SUBST (Θ_2, SUBST (Θ_1, p))
- For Example:-

  P – Sells (x,M1, y)

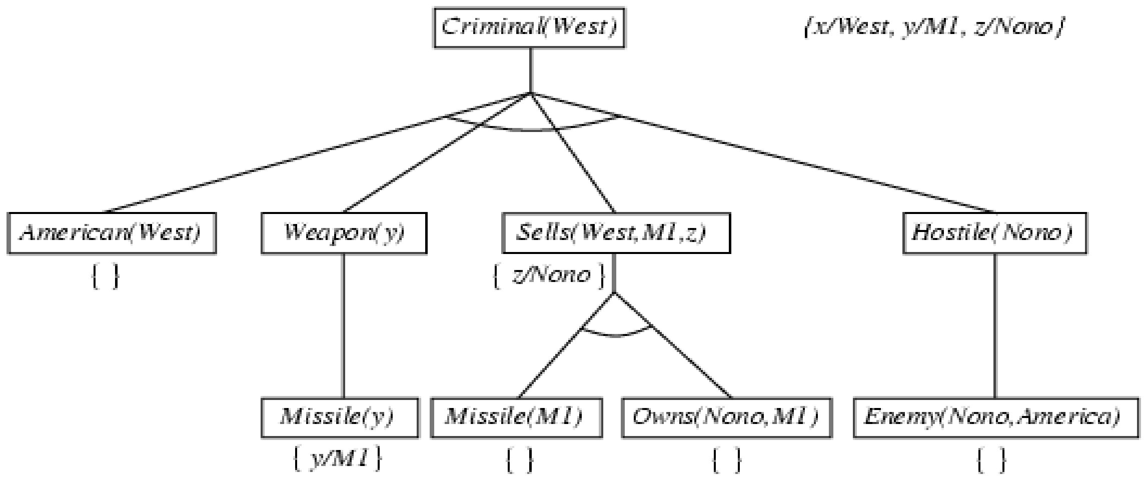
SUBST (Θ_2, SUBST (Θ_1, p))
SUBST (Θ_2, SUBST (x/West, p)) i.e. (Θ_1 = x/west)
SUBST ((y/Nono), (x/West, p)) i.e. (Θ_2 = y/Nono)
Therefore p – Sells (West, M1, Nono)

- The following table shows the backward chaining algorithm,

```
function FOL-BC-ASK(KB, goals, θ) returns a set of substitutions
  inputs: KB, a knowledge base
          goals, a list of conjuncts forming a query
          θ, the current substitution, initially the empty substitution { }
  local variables: ans, a set of substitutions, initially empty

  if goals is empty then return {θ}
  q′ ← SUBST(θ, FIRST(goals))
  for each r in KB where STANDARDIZE-APART(r) = ( p1 ∧ ... ∧ pn ⇒ q)
          and θ′ ← UNIFY(q, q′) succeeds
    ans ← FOL-BC-ASK(KB, [p1, ..., pn|REST(goals)], COMPOSE(θ, θ′)) ∪ ans
  return ans
```

- The following graph shows the proof tree to infer that west is a criminal,



- To prove Criminal(x), we have to prove the five conjuncts below it
- Some of which are directly exist in the knowledge base, others require one more iteration of backward chaining.
- In the search process the substitution of values for the variables has to be done in a correct way, otherwise it may lead to failure solution.
- The following are the some properties of Backward Chaining,
  - Depth-first recursive proof search: space is linear in size of proof
  - Incomplete due to infinite loops
    - ⇒ fix by checking current goal against every goal on stack

- Inefficient due to repeated subgoals (both success and failure)
  - $\Rightarrow$ fix using caching of previous results (extra space)
- Widely used for logic programming

2.8.1 Logic Programming:-

- A system in which KB can be constructed and used.
- A relation between logic and algorithm is summed up in Robert Kowalsh equation

  Algorithm = Logic + Control
- Logic programming languages, usually use backward chaining and input/output of programming languages.
- A logic programming language makes it possible to write algorithms by augmenting logic sentences with information to control the inference process.
- For Example:- PROLOG
  - ✓ A prolog program is described as a series of logical assertions each of which is a Horn Clause.
  - ✓ A Horn Clause is a Clause that has atmost one positive literal,

    Example: - P, $\neg P \wedge Q$
  - ✓ Implementation: - All inferences are done by backward chaining, with depth first search. The order of search through the conjuncts of an antecedent is left to right and the clauses in the KB are applied first-to-last order.
- **Example for FOL to PROLOG conversion:-**
  - **FOL**
    - ✓ **$\forall x$ Pet(x) $\wedge$ Small(x) $\Rightarrow$ Apartment(x)**
    - ✓ **$\forall x$ Cat(x) v Dog(x) $\Rightarrow$ Pet(x)**
    - ✓ **$\forall x$ Product(x) $\Rightarrow$ Dog(x) $\wedge$ Small(x)**
    - ✓ **Poodle(fluffy)**
  - **Equivalent PROLOG representation**
    - ✓ **Apartment(x) :- Pet(x), Small(x)**
    - ✓ **Pet(x) :- Cat(x)**
      Pet(x) :- Dog(x)
    - ✓ **Dog(x) :- Poodle(x)**
      Small(x) :- Poodle(x)
    - ✓ **Poodle(fluffy)**
  - In the PROLOG representation the consequent or the left hand side is called as head and the antecedent or the right hand side is called as **body.**
- **Execution of a PROLOG program:-**
  - The execution of a prolog program can happen in two modes,
    1. Interpreters
    2. Compilers
  - **Interpretation:**
    - ✓ A method which uses **BACK-CHAIN** algorithm with the program as the **KB.**

- ✓ To maximize the speed of execution, we will consider two different types of constraints executed in sequence, They are
  1. **Choice Point: -** Generating sub goals one by one to perform interpretation.
  2. **Trail: -** Keeping track of all the variables that have been bound in a stack is called as trail.
- o **Compilation:-**
  - ✓ Procedure implementation is done to run the program (i.e.) calling the inference rules whenever it is required for execution.

2.9 Unification:-

- The process of finding all legal substitutions that make logical expressions look identical and
- Unification is a "pattern matching" procedure that takes two atomic sentences, called literals, as input, and returns "failure" if they do not match and a substitution list, Theta, if they do match.
- Theta is called the most general unifier (mgu)
- All variables in the given two literals are implicitly universally quantified
- To make literals match, replace (universally-quantified) variables by terms
- The unification routine, UNIFY is to take two atomic sentences p and q and returns α substitution that would make p and q look the same

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

  θ = Unifier of two sentences
- For example:-

  p – S1(x, x) q

  – S1(y, z)

  Assume θ = y

  p – S1(y, y) – x/y (Substituting y for x)

  q – S1(y, y) – z/y (Substituting y for z)
- In the above two sentences (p, q), the unifier of two sentences (i.e.) θ = y is substituted in both the sentences, which derives a same predicate name, same number of arguments and same argument names.
- Therefore the given two sentences are unified sentences.
- The function UNIFY will return its result as fail, for two different types of criteria's as follows,
  - ✓ If the given two atomic sentences (p, q) are differs in its predicate name then the UNIFY will return failure as a result

    For Example: - p – hate (M, C), q – try (M, C)
  - ✓ If the given two atomic sentences (p, q) are differs in its number of arguments then the UNIFY will return failure as a result

    For Example: - p – try (M, C), q – try (M, C, R)
- For Example: - The Query is **Knows (John, x) whom does John Know?**

- Some answers to the above query can be found by finding all sentences in the KB that unify with knows (John, x)
- Assume the KB has as follows,
  Knows (John, John)
  Knows (y, Leo)
  Knows (y, Mother(y))
  Knows (x, Elizabeth)
- The results of unification are as follows,
  UNIFY (knows (john, x), knows (John, Jane)) = {x/Jane}
  UNIFY (knows (john, x), knows (y, Leo)) = {x/Leo, y/John}
  UNIFY (knows (john, x), knows (y, Mother(y)) = {y/John, x/Mother (John)}
  UNIFY (knows (john, x), knows (x, Elizabeth)) = fail
- x cannot take both the values John and Elizabeth at the same time.
- Knows (x, Elizabeth) means "Everyone knows Elizabeth" from this we able to infer that John knows Elizabeth.
- This can be avoided by using standardizing apart one of the two sentences being unified (i.e.) renaming is done to avoid name clashes.
- For Example:-
  UNIFY (Knows (john, x), knows (x_1, Elizabeth)) = {x/Elizabeth, x_1/John}

2.9.1 Most general Unifier (MGU):-

- UNIFY should return a substitution that makes the two arguments look the same, but there may be a chance of more than one unifier.
- For Example:-
  UNIFY (knows (john, x), knows (y, z)) = {y/John, x/z} or {y/John, x/John, z/John}
- The result of applying 1st unifier is knows (John, z) and the 2nd unifier is knows (John, John).
- Here the first unifier result is more general than the second one, because it places less restriction on the values of the variables.
- This indicates that every unifiable pair of expressions, a single MGU is exist until renaming of variables.
- The following table shows the unification algorithm,
- The following are the steps to be done for unification of two sentences p and q is given below,
  - ✓ Recursively explore the two expressions simultaneously along with unifier returns failure if two corresponding points in the structure do not match. Therefore the time complexit y is O(n^2) in the size of the expressions being unified.
  - ✓ When the variable is matched against a complex term, one must check, whether the variable itself occurs, if it is true then returns failure (consistent unifier is not allowed) – occur check.

```
function UNIFY($x, y, \theta$) returns a substitution to make $x$ and $y$ identical
   inputs: $x$, a variable, constant, list, or compound
           $y$, a variable, constant, list, or compound
           $\theta$, the substitution built up so far

   if $\theta$ = failure then return failure
   else if $x = y$ then return $\theta$
   else if VARIABLE?($x$) then return UNIFY-VAR($x, y, \theta$)
   else if VARIABLE?($y$) then return UNIFY-VAR($y, x, \theta$)
   else if COMPOUND?($x$) and COMPOUND?($y$) then
       return UNIFY(ARGS[$x$], ARGS[$y$], UNIFY(OP[$x$], OP[$y$], $\theta$))
   else if LIST?($x$) and LIST?($y$) then
       return UNIFY(REST[$x$], REST[$y$], UNIFY(FIRST[$x$], FIRST[$y$], $\theta$))
   else return failure
```

```
function UNIFY-VAR($var, x, \theta$) returns a substitution
   inputs: $var$, a variable
           $x$, any expression
           $\theta$, the substitution built up so far

   if $\{var/val\} \in \theta$ then return UNIFY($val, x, \theta$)
   else if $\{x/val\} \in \theta$ then return UNIFY($var, val, \theta$)
   else if OCCUR-CHECK?($var, x$) then return failure
   else return add $\{var/x\}$ to $\theta$
```

- The above table shows the unification algorithm

2.9.2 Storage and retrieval:-

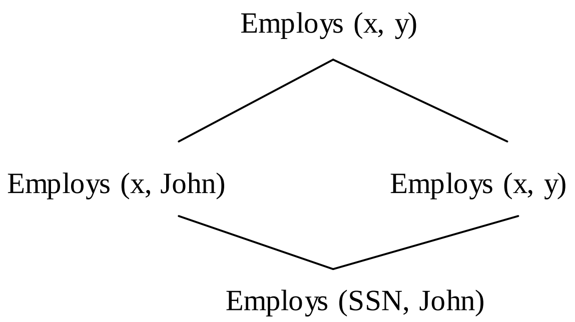
- Once the data type for sentences and terms are defined, we need to maintain asset of sentences in a KB (i.e.) to store and fetch a sentence or term,
    - ✓ Store (s) – stores a sentence s.
    - ✓ Fetch (q) – returns all unifiers
- Such that the query q unifies with some sentences in the KB.
- For Example: - The unification for Knows (John, x) is an Instance of fetching.
- The simplest way to store and fetch is maintain a long list in sequential order.
- For a Query q, call UNIFY (q, s) for every sentences s in the list, requires O(n) time on an n-element KB.
- To make the fetch more efficient, indexing the facts in KB is done.
- The different types of indexing are as follows,
    - ✓ Table based Indexing
    - ✓ Tree based Indexing

- ✓ Predicate based Indexing

- A simple form of indexing is predicate indexing puts all the knows facts in one bucket and all the brother facts in another.
- The buckets are stored in hash table for efficient access,
- Where hash table means "a data structure for storing and retrieving information indexed by fixed keys"
- Given sentence to be stored, it is possible to construct indices for all possible queries that unify with it,
- For Example:- For the fact Employs (SSN, John) the queries are as follows,

✓ Employs (SSN, John)	Does SSN employ John?
✓ Employs (x, John)	who employs John?
✓ Employs (SSN, y)	whom does SSN employ?
✓ Employs (x, y)	who employs whom?

- These queries form a substitution lattice (i.e.) Properties of Lattice:- child of any node in the lattice is obtained from its parent by a single substitution and the highest common descendent of any two nodes is the result of applying their most general unifier.
- For a predicate with n arguments, the lattice contains O(2^n) nodes
- The following diagram shows the subsumption lattice,



2.9.3 Advantages and Disadvantages:-

Advantages:-

- The scheme works very well whenever the lattice contains a small number of nodes.
- For a predicate with n arguments, the lattice contains O(2n) nodes.

Disadvantages:-

- If function symbols are allowed, the number of nodes is also exponential in the size of the terms in the sentence to be stored. This can lead to a huge number of indices.
- At some point, the benefits of indexing are outweighed by the costs of storing and maintaining all the indices.

2.10 Resolution:-

- Resolution is a complete inference procedure for first order logic
- Any sentence a entailed by KB can be derived with resolution
- Catch: proof procedure can run for an unspecified amount of time

  - At any given moment, if proof is not done, don't know if infinitely looping or about to give an answer
  - Cannot always prove that a sentence a is *not* entailed by KB
  - First-order logic is semidecidable
- Rules used in the resolution procedure are :
  - Simple resolution inference rule – premises have exactly two disjuncts

$$\frac{\alpha \vee \beta, \neg\beta \vee \gamma}{\alpha \vee \gamma} \quad \text{or equivalently} \quad \frac{\neg\alpha \Rightarrow \beta, \beta \Rightarrow \gamma}{\neg\alpha \Rightarrow \gamma}$$

  - Resolution inference rule – two disjunctions of any length, if one of disjunct in the class (p_j) unifies with the negation of the disjunct in the other class, then infer the disjunction of all the disjuncts except for those two,
  - **Using disjunctions:-** For literals p_i and q_i, where UNIFY (p_j, ¬q_k) = θ

$$\frac{p_1 \vee \ldots\ldots\ldots\ldots p_j \ldots\ldots\ldots\ldots \vee\; p_m,\; q_1 \vee \ldots\ldots\ldots\ldots q_k \ldots\ldots\ldots \vee\; q_n}{SUBST(\theta,(p_1 \vee \ldots\ldots\ldots\ldots p_{j-1} \vee p_{j+1} \ldots\ldots\ldots p_m \vee q_1 \ldots\ldots\ldots\ldots q_{k-1} \vee q_{k+1} \ldots\ldots\ldots\ldots \vee q_n}$$

  - **Using implications:-** For atoms p_i, q_i, r_i, s_i where UNIFY (p_j, q_k) = θ

$$\frac{\begin{array}{c} p_1 \wedge \ldots\ldots\ldots\ldots p_j \wedge \ldots\ldots\ldots\ldots p_{n1} \Rightarrow r_1 \vee \ldots\ldots\ldots r_{n2} \\ \\ s_1 \wedge \ldots\ldots\ldots\ldots s_{n3} \Rightarrow, q_1 \vee \ldots\ldots\ldots\ldots q_k \ldots\ldots\ldots q_{n4} \end{array}}{SUBST(\theta,(p_1 \wedge \ldots p_{j-1} \wedge p_{j+1} \ldots\ldots p_{n1} \wedge s_1 \wedge \ldots\ldots s_{n3} \Rightarrow r_1 \vee \ldots\ldots\ldots r_{n2} \vee q_1 \vee \ldots\ldots\ldots\ldots q_{k-1} \vee q_{k+1} \ldots\ldots\ldots\ldots \vee q_{n4}}$$

2.10.1 Canonical Form (or) Normal form for Resolution:-

- The canonical form representation of sentences for resolution procedure (to derive pa proof) is done in two ways, they are as follows,
  - **Conjunctive Normal form (CNF):-** All the disjunctions are joined aqs one big sentences.
  - **Implicative Normal Form (INF):-** All the conjunctions of atoms on the left and a disjunction of atoms on the right.
- The following table shows the Knowledge base for CNF and INF,

Conjuctive Normal Form	Implicative Normal Form
¬ P(w) ∨ Q(w)	P(w) ⇒ Q(w)
P(x) ∨ R(x)	True ⇒ P(x) ∨ R(x)
¬ Q(y) ∨ S(y)	Q(y) ⇒ S(y)
¬ R(z) ∨ S(z)	R(z) ⇒ S(z)

- Resolution is a generalization of Modus Ponen.
- The following is the representation of Modus Ponen rule in resolution as a special case (i.e.),

$$\frac{\alpha, \alpha \Rightarrow \beta}{\beta} \quad \text{is equivalent to} \quad \frac{True \Rightarrow \alpha, \alpha \Rightarrow \beta}{True \Rightarrow \beta}$$

2.10.2 Methods used for resolution technique

- **Resolution Proof:-** A set of inference rules and resolution rules can be used to derive a conclusion from the KB.
- **Resolution with refutation (or) proof by contradiction (or) reduction and absurdum:-** One complete inference procedure using resolution is refutation. The idea is to prove P, we assume P is false (i.e. add $\neg$P to KB) and prove a contradiction, that is the KB implies P.

$$\textbf{(KB} \wedge \neg \textbf{P} \Rightarrow \textbf{False)} \Leftrightarrow \textbf{(KB} \Rightarrow \textbf{P)}$$

- **Example:**

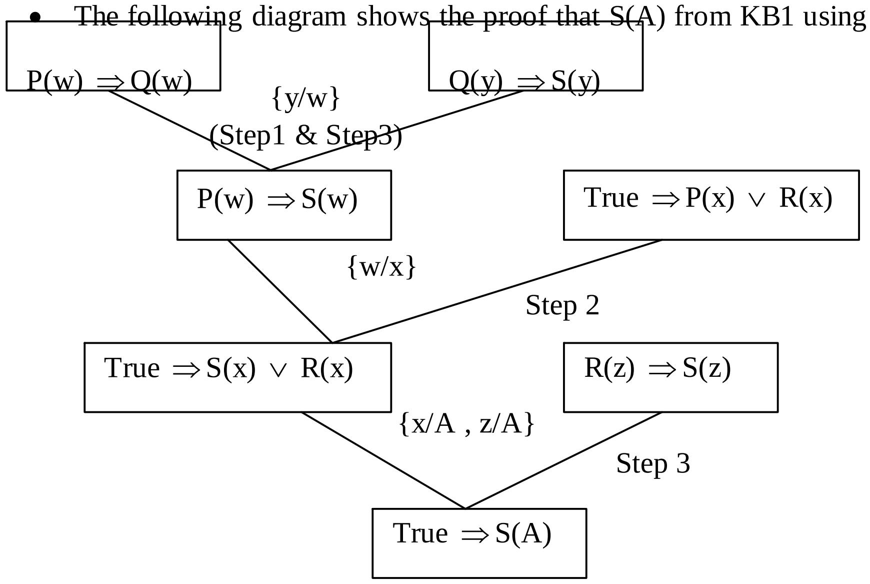
1. Prove that S(A) ts true from KB1 of CNF and INF representation using
   - ✓ Resolution Proof
   - ✓ Resolution with refutation

(a) Resolution Using INF representation:-

Given (KB1):-

1. P(w) $\Rightarrow$ Q(w)
2. True $\Rightarrow$ P(x) $\vee$ R(x)
3. Q(y) $\Rightarrow$ S(y)
4. R(z) $\Rightarrow$ S(z)

- The following diagram shows the proof that S(A) from KB1 using resolution



- Resolution rule:- In the first step transitive implication rule is used
- Substitution of one predicate in the other. (i.e.) P(x) $\Rightarrow$ S(x) is substituted in True $\Rightarrow$ P(x) $\vee$ S(x) that is instead of P(x), S(x) is substituted.

- Substitution of one predicate in the other. (i.e.) R(A) $\Rightarrow$S(A) is substituted in True $\Rightarrow$S(A) $\vee$ R(A) that is instead of R(A), S(A) is substituted, which derives True $\Rightarrow$S(A) $\vee$ S(A) is equivalent to True $\Rightarrow$S(A)
- Therefore S(A) is true is proved using resolution technique for INF representation.
- Where each "Vee" Shape in the proof tree represents a resolution step,
- The two sentences at the top are the premises from the KB, and the one at the bottom is the conclusion (or) resolvent.

b. Resolution Using CNF representation:-
Given (KB1):-

1. ~P(w) $\vee$ Q(w)
2. P(x) $\vee$ R(x)
3. ~Q(y) $\vee$ S(y)
4. ~R(z) $\vee$ S(z)

- **Resolution rule:-**

$$\frac{\alpha\vee\beta,\ \beta\vee\gamma}{\alpha\vee\gamma} \quad \text{(i.e.)} \quad \frac{\sim P(w)\vee Q(w), \sim Q(w)\vee S(w)}{\sim P(w)\vee S(w)}$$

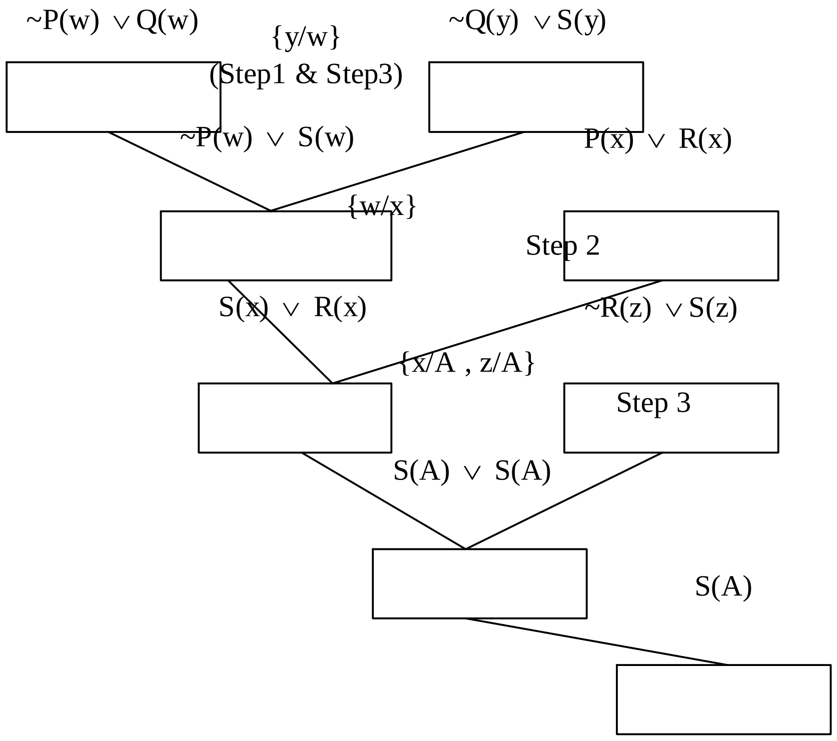
- **Resolution rule:-**

$$\frac{\alpha\vee\beta,\ \beta\vee\gamma}{\alpha\vee\gamma} \quad \text{(i.e.)} \quad \frac{R(x)\vee P(x), \sim P(x)\vee S(x)}{R(x)\vee S(x)}$$

- **Resolution rule:-**

$$\frac{\alpha\vee\beta,\ \beta\vee\gamma}{\alpha\vee\gamma} \quad \text{(i.e.)} \quad \frac{S(A)\vee R(A), \sim R(A)\vee S(A)}{S(A)\vee S(A)}$$

- Therefore S(A) is true is proved using resolution technique for CNF representation.
- The following diagram shows the proof that S(A) from KB1 using resolution.



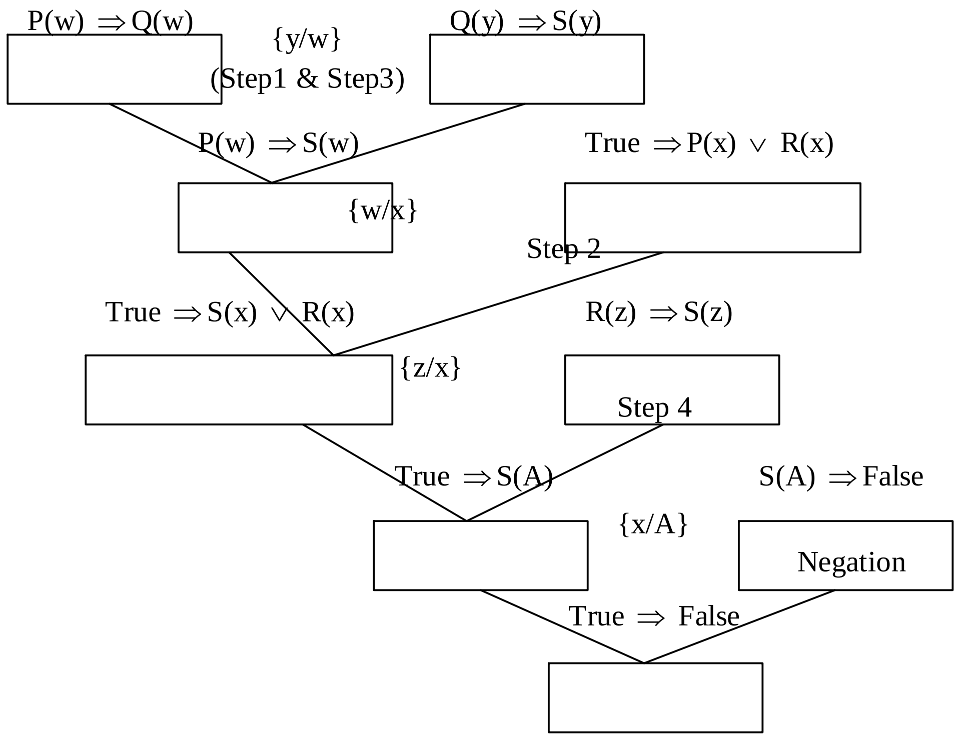
C. Resolution with refutation Proof using INF representation:-
Given (KB1):-

1. $P(w) \Rightarrow Q(w)$
2. $True \Rightarrow P(x) \vee R(x)$
3. $Q(y) \Rightarrow S(y)$
4. $R(z) \Rightarrow S(z)$

- **Resolution rule:-** In this step transitive implication is applied,

$$\frac{\alpha \Rightarrow \beta, \beta \Rightarrow \gamma}{\alpha \Rightarrow \gamma}$$

- Substitution of one predicate in the other. (i.e.) $P(x) \Rightarrow S(x)$ is substituted in True $\Rightarrow P(x) \vee S(x)$ that is instead of P(x), S(x) is substituted.
- Substitution of one predicate in the other. (i.e.) $R(x) \Rightarrow S(x)$ is substituted in True $\Rightarrow S(x) \vee R(x)$ that is instead of R(x), S(x) is substituted, which derives True $\Rightarrow S(x) \vee S(x)$ is equivalent to True $\Rightarrow S(x)$
- To prove using refutation, negation of given proof is added to the KB and derives a contradiction, then the given statement is true otherwise it is false.
- Therefore in step 4, we assume $S(A) \Rightarrow$ False, derives a contradiction True $\Rightarrow$ False.
- Therefore S(A) is true is proved using refutation technique in INF representation.
- The following diagram shows the proof that S(A) from KB1 using resolution with refutation in INF representation.



D. Resolution with refutation using CNF representation:-
Given KB1:-

1. ~P(w) $\vee$ Q(w)
2. P(x) $\vee$ R(x)
3. ~Q(y) $\vee$ S(y)
4. ~R(z) $\vee$ S(z)

- **Resolution rule:-**

$$\frac{\alpha\vee\beta,\ \beta\vee\gamma}{\alpha\vee\gamma} \quad \text{(i.e.)} \quad \frac{\sim P(w)\vee Q(w),\sim Q(w)\vee S(w)}{\sim P(w)\vee S(w)}$$

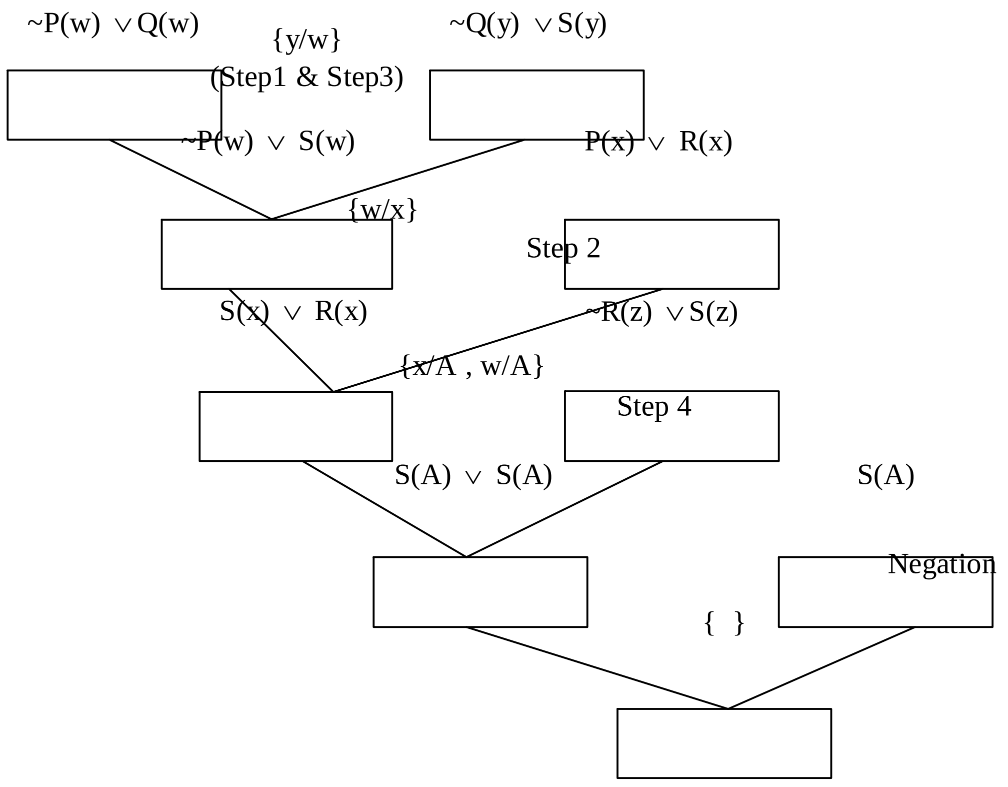
- **Resolution rule:-**

$$\frac{\alpha\vee\beta,\ \beta\vee\gamma}{\alpha\vee\gamma} \quad \text{(i.e.)} \quad \frac{R(x)\vee P(x),\sim P(x)\vee S(x)}{R(x)\vee S(x)}$$

- **Resolution rule:-**

$$\frac{\alpha\vee\beta,\ \beta\vee\gamma}{\alpha\vee\gamma} \quad \text{(i.e.)} \quad \frac{S(A)\vee R(A),\sim R(A)\vee S(A)}{S(A)\vee S(A)}$$

- To prove using refutation, the negation of the given statement to be proved is added to the KB and it derives a empty set, represents that the statement is True in the KB.
- Therefore S(A) is True is proved using resolution with refutation in CNF representation.
- The following diagram shows the proof that S(A) from KB1 using resolution with refutation in CNF representation.



2.10. 3 Resolution technique (From FOL Representation):-

- In the previous section we learned how to prove the given fact using two different types of resolution techniques (Resolution proof, Resolution with refutation proof) from the given KB representation (CNF, INF).
- Suppose if the KB is given as a English description, to prove a fact , then how to derive the conclusion? What are the sequence of steps to be done? Are discussed one by one,
  - The given KB description is converted into FOL Sentences
  - FOL sentences are converted to INF (or) CNF representation without changing the meaning of it (using conversion to normal form procedure)
  - Apply one of the resolution technique (Resolution proof (or) Resolution with refutation proof), to resolve the conflict.
  - Derive the fact to be proved or declare it as a incomplete solution.

2.10.3.1 Conversion to Normal Form or Clause Form (Procedure) :-

1. **Eliminate Implications:-**
   Eliminate Implication by the corresponding disjunctions,
   (i.e.) $p \Rightarrow q$ is the same as $\neg p \vee q$
2. **Move $\neg$ inwards :-**
   Negations are allowed only on atoms in CNF and not at all in INF. Therefore eliminate negations with, Demorgan's laws, the quantifier equivalences and double negation.
   Demorgans Law
   $\neg(p \vee q)$ becomes $\neg p \wedge \neg q$
   $\neg(p \wedge q)$ becomes $\neg p \vee \neg q$
   Quantifier equivalences
   $\neg \forall xp$ becomes $\exists x \neg p$
   $\neg \exists xp$ becomes $\forall x \neg p$
   Double Negation
   $\neg\neg p$ becomes p Double negation
3. **Standardize variables:-**
   If the sentence consists of same variable name twice, change the name of one of the variable. This avoids confusion later when we drop the quantifiers,
   (i.e.) $(\forall xp(x)) \vee (\exists xQ(x))$ is changed into $(\forall xp(x)) \vee (\exists yQ(y))$
4. **Move quantifiers left:-**
   The quantifiers in the middle of the sentence to be moved to the left, without changing the meaning of the sentence
   (i.e.) $p \vee \forall$ xq becomes $\forall x \quad p \vee q$, which is true because p here is guaranteed not to contain x.
5. **Skolimize:-**
   Skolimization is the process of removing existential quantifiers by elimination, it is very similar to the existential elimination rule.
   (i.e.) $\exists xp(x)$ into p(A), where A is a constant that does not appear elsewhere in the KB.

For Example:- "Everyone has a heart"
- FOL : $\forall x$ person(x) $\Rightarrow$ $\exists y$ Heart (y) $\wedge$ Has (x, y)
- Replace $\exists y......y$ with a constant H

$\forall x$ person (x) $\Rightarrow$ Heart (H) $\wedge$ Has (x, y)

- The above representation says that everyone has the same heart H, not necessarily shared between each other. It can be rewritten by applying a function to each person that maps from person to heart.

  $\forall x$ person (x) $\Rightarrow$ Heart (F(x) $\wedge$ Has (x, F(x)) where F is a function that does not appear elsewhere in the KB and it is called as Skolem function.

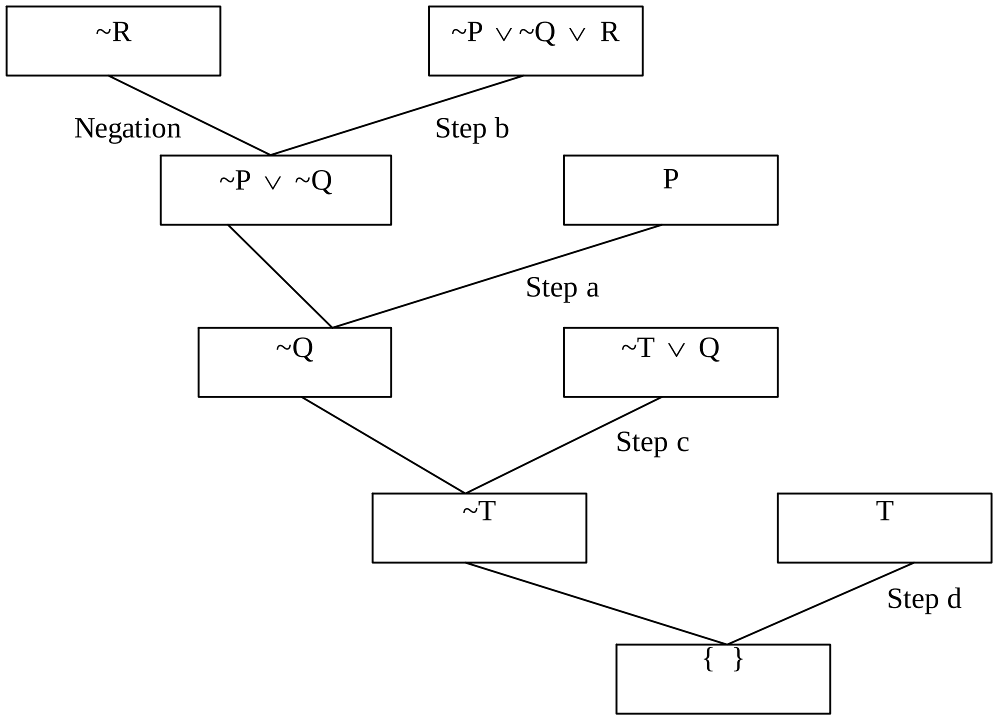
6. **Distribute $\wedge$ over $\vee$:-**
   $(a \wedge b) \vee c$ becomes $(a \vee c) \wedge (b \vee c)$
7. **Flatten nested conjunctions and disjunctions :-**
   $(a \vee b) \vee c$ becomes $(a \vee b \vee c)$

   $(a \wedge b) \wedge c$ becomes $(a \wedge b \wedge c)$

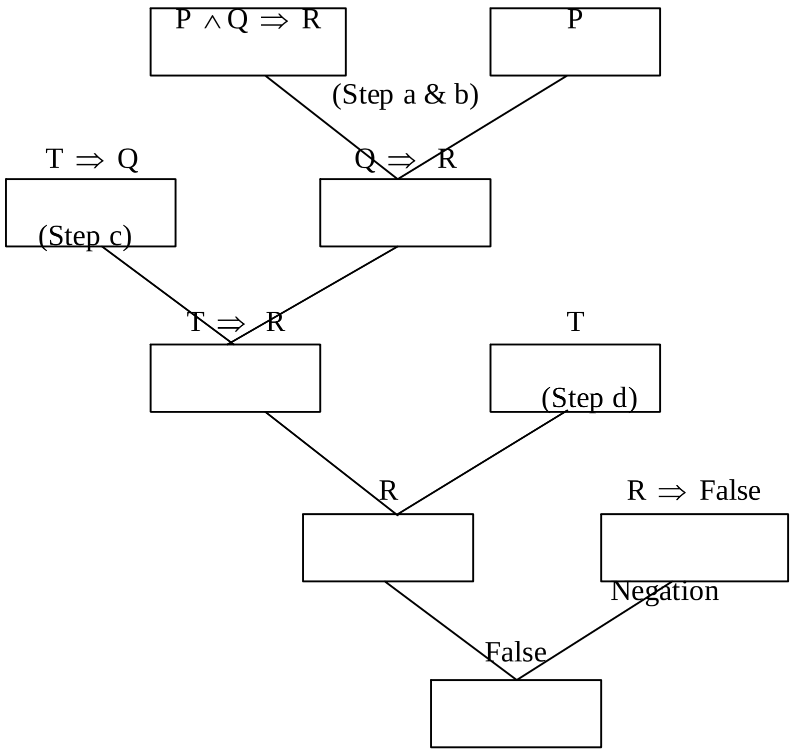
   It is a conjunction where every conjunct is a disjunction of literals.
8. **Convert disjunctions to implications :-**
   From the CNF it is possible to derive the INF, (i.e.) combine the negative literals into one list, the positive literals into another and build an implication from them,

   (i.e.) $(\neg a \vee \neg b \vee c \vee d)$ becomes $(a \wedge b \Rightarrow c \vee d)$

- 1**. For Example: -** Covert the given axioms into equivalent clasuses form and prove that R is true using CNF and INF representation.
  - Axioms:
    - P
    - P $\wedge$ Q $\Rightarrow$R
    - S $\vee$ T $\Rightarrow$ Q
    - T
  - Proof:
    - ✓ Equivalent Conjunctive Clause Form Representation:
      - P
      - $\neg$P $\vee$ $\neg$Q $\vee$ R
      - $\neg$S $\vee$ Q, $\neg$T $\vee$ Q
      - T
    - ✓ Equivalent INF Representation:
      - P
      - P $\wedge$ Q $\Rightarrow$R
      - S $\Rightarrow$ Q, T $\Rightarrow$ Q
      - T
- The negation of the given statement (~R) derives a contradiction ({ }).
- Therefore it is proved that R is true.
- The following diagram shows the proof of resolution with refutation using CNF.

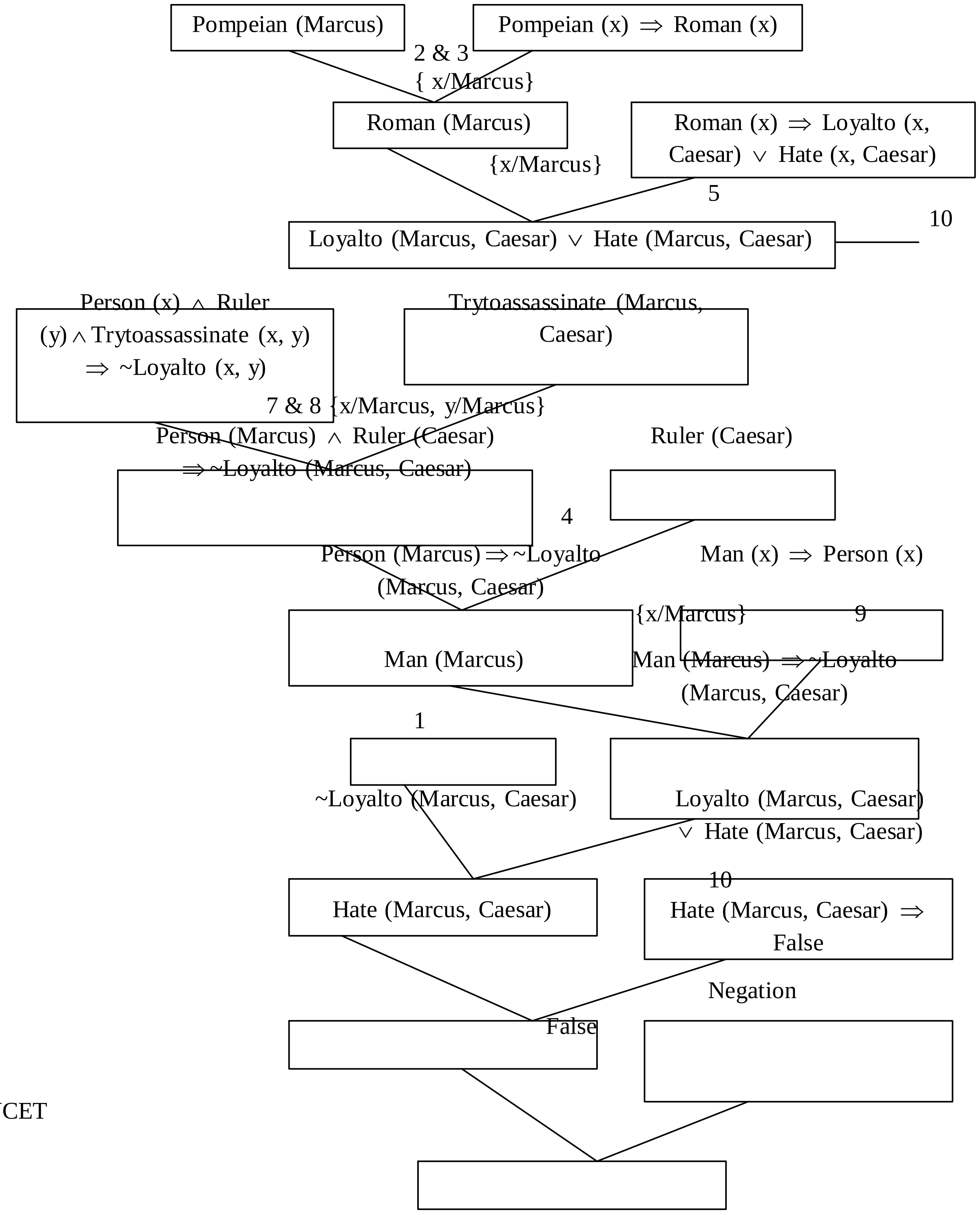


- The negation of the given statements (R ⇒ False) derives a contradiction (True ⇒ False).
- Therefore it is proved that R is True (i.e. Taken assumption is wrong)
- The following diagram shows the proof of Resolution with refutation using INF.

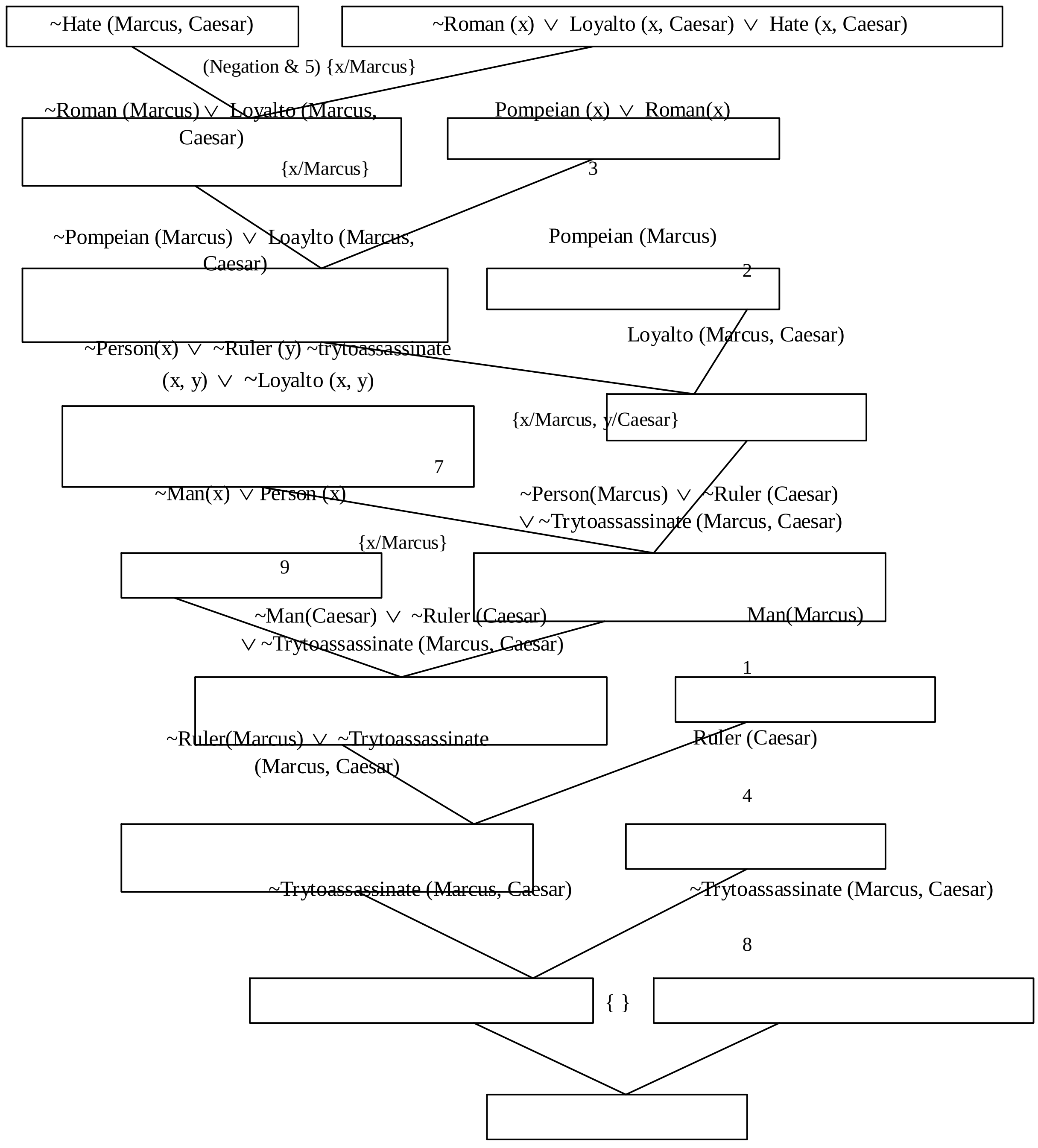


- **2. For Example: -** Convert the given sentences (KB2) into its equivalent FOL representation and then convert it into its CNF and INF representation and solve the task using resolution with refutation proof.
  - Given KB2:-
    - Marcus was a man
    - Marcus was a Pompeian
    - All pompeians were romans
    - Caesar was a ruler
    - All romans were either loyalto Caesar or hated him
    - Everyone is loyal to someone
    - People only try to assassinate rulers they are not loyal to.
    - Marcus tried to assassinate Caesar
    - All man are people
  - **Task: -** Did Marcus hate Caesar?
  - **Solution:-**
    - ✓ **FOL Representation:**
      1. Man (Marcus)
      2. Pompeian (Marcus)
      3. $\forall x$ Pompeian (x) $\Rightarrow$ Roman (x)
      4. Ruler (Caesar)
      5. $\forall x$ Roman (x) $\Rightarrow$ Loyalto (x, Caesar) $\vee$ Hate (x, Caesar)
      6. $\forall x\ \exists y$ Loyalto (x, y)
      7. $\forall x\ \forall \mathbf{y}$ Person(x) $\wedge$ Ruler(y) $\wedge$ Trytoassassinate(x, y) $\Rightarrow \neg$ Loyalto(x, y)
      8. Trytoassassinate (Marcus, Caesar)
      9. $\forall x$ Man (x) $\Rightarrow$ Person (x)
    - ✓ **INF Representation:**
      1. Man (Marcus)
      2. Pompeian (Marcus)
      3. Pompeian (x) $\Rightarrow$ Roman (x)
      4. Ruler (Caesar)
      5. Roman (x) $\Rightarrow$ Loyalto (x, Caesar) $\vee$ Hate (x, Caesar)
      6. Loyalto (x, y)
      7. Person(x) $\wedge$ Ruler(y) $\wedge$ Trytoassassinate(x, y) $\Rightarrow \neg$Loyalto(x, y)
      8. Trytoassassinate (Marcus, Caesar)
      9. Man (x) $\Rightarrow$ Person (x)
    - ✓ **CNF Representation :**
      1. Man (Marcus)
      2. Pompeian (Marcus)
      3. ~Pompeian (x) $\vee$ Roman (x)
      4. Ruler (Caesar)
      5. ~Roman (x) $\vee$ Loyalto (x, Caesar) $\vee$ Hate (x, Caesar)
      6. Loyalto (x, y)
      7. ~Person(x) $\vee$ ~Ruler(y) $\vee$ ~Trytoassassinate(x, y) $\vee$ ~Loyalto(x, y)
      8. Trytoassassinate (Marcus, Caesar)
      9. ~Man (x) $\vee$ Person (x)

- To prove the given fact, we assumed the negation of it (i.e.) Hate(Marcus, Caesar) ⇒ False and using inference rules we derive a contradiction False, which means that the assumption must be false, and Hate(Marcus, Caesar) is True.
- Therefore it is proved that Hate (Marcus, Caesar) is True using INF Representation.
- The following diagram shows the proof of Resolution with refutation using INF representation.


Pompeian (Marcus)
Pompeian (x) ⇒ Roman (x)
2 & 3
{ x/Marcus}
Roman (Marcus)
Roman (x) ⇒ Loyalto (x, Caesar) ∨ Hate (x, Caesar)
{x/Marcus}
5
10
Loyalto (Marcus, Caesar) ∨ Hate (Marcus, Caesar)
Person (x) ∧ Ruler (y) ∧ Trytoassassinate (x, y) ⇒ ~Loyalto (x, y)
Trytoassassinate (Marcus, Caesar)
7 & 8 {x/Marcus, y/Marcus}
Person (Marcus) ∧ Ruler (Caesar) ⇒ ~Loyalto (Marcus, Caesar)
Ruler (Caesar)
4
Person (Marcus) ⇒ ~Loyalto (Marcus, Caesar)
Man (x) ⇒ Person (x)
{x/Marcus}
9
Man (Marcus)
Man (Marcus) ⇒ ~Loyalto (Marcus, Caesar)
1
~Loyalto (Marcus, Caesar)
Loyalto (Marcus, Caesar) ∨ Hate (Marcus, Caesar)
10
Hate (Marcus, Caesar)
Hate (Marcus, Caesar) ⇒ False
Negation
False

- To prove the given fact, we assumed that the negation of it (i.e.) ~ Hate (Marcus, Caesar) and using inference rules, we derive a contradiction as empty set, which means that the assumption must be false, and Hate (Marcus, Caesar) is true in the given KB.
- Therefore it is proved that Hate (Marcus, Caesar) is true using CNF representation.
- The following diagram shows the proof of Resolution with refutation using CNF representation.


~Hate (Marcus, Caesar)
~Roman (x) ∨ Loyalto (x, Caesar) ∨ Hate (x, Caesar)
(Negation & 5) {x/Marcus}
~Roman (Marcus) ∨ Loyalto (Marcus, Caesar)
Pompeian (x) ∨ Roman(x)
{x/Marcus}
3
~Pompeian (Marcus) ∨ Loaylto (Marcus, Caesar)
Pompeian (Marcus)
2
Loyalto (Marcus, Caesar)
~Person(x) ∨ ~Ruler (y) ~trytoassassinate (x, y) ∨ ~Loyalto (x, y)
{x/Marcus, y/Caesar}
7
~Man(x) ∨ Person (x)
~Person(Marcus) ∨ ~Ruler (Caesar) ∨ ~Trytoassassinate (Marcus, Caesar)
{x/Marcus}
9
~Man(Caesar) ∨ ~Ruler (Caesar) ∨ ~Trytoassassinate (Marcus, Caesar)
Man(Marcus)
1
~Ruler(Marcus) ∨ ~Trytoassassinate (Marcus, Caesar)
Ruler (Caesar)
4
~Trytoassassinate (Marcus, Caesar)
~Trytoassassinate (Marcus, Caesar)
8
{ }

.10.4 Dealing with equality:-

- The unification algorithm is used to find a equality between variables with other terms (i.e.) p(x) unifies with p(A), but p(A) and p(B) fails to unify, even if the sentence A=B in the KB.

- The unification does only a syntactic test based on the appearance of the argument terms, not a true semantic test (meaning) based on the objects they represent.

- To solve this problem, one of the way is, the properties can be followed as,

1.	$\forall x$	$x = x$	- Reflexive
2.	$\forall x, y$	$x = y \Rightarrow y = x$	- Symmetric
3.	$\forall x, y, z$	$x = y \wedge y = z \Rightarrow x = z$	- Transitive
4.	$\forall x, y$	$x = y \Rightarrow (p_1(x) \Leftrightarrow p_1(y))$	$x = y$, if the predicate name
	$\forall x, y$	$x = y \Rightarrow (p_2(x) \Leftrightarrow p_2(y))$	and arguments are same.
5.	$\forall w, x, y, z$	$w = y \wedge x = z \Rightarrow (F_1(w, x) = F_1(y, z))$	
		$w = y \wedge x = z \Rightarrow (F_2(w, x) = F_2(y, z))$	

- The other way to deal with equality is **demodulation rule**. For any terms x, y and z where UNIFY (x, z) = θ, defined as :

$$\frac{x = y, (......z......)}{(.....SUBST\,(\theta, y).....)}$$

- A more powerful rule named paramodulation deals with the case where we do not know $x = y$, but we do know $x = y \vee p(x)$.

2.10.5 Resolution strategies:-

- A strategy which is used to guide the search towards a proof of the resolution inference rule. Different types of strategies are as follows,

✓ **Unit Preferences:-**

This strategy prefers a sentence with single literal, to produce a very short sentence

For Example:- P and $P \wedge Q \Rightarrow R$ derives the sentence $Q \Rightarrow R$.

✓ **Set of support:-**

A subset of the sentence are identified (set of support) and resolution combines a set of support with another sentence and adds the resolvent into the set off support, which reduces the size of a sentence in the KB.

Disadvantage: - Bad choice for the set of support will make the algorithm incomplete.

Advantage: - Goal directed proof trees are generated

- ✓ **Input resolution:-**

  This strategy combines one of the input sentences (from the KB or the query) with some other sentence.

  For Example:- Resolution proof, Resolution with refutation proof.

  Input resolution is complete for KB that are in Horn form, but incomplete in the general case.

- ✓ **Linear resolution:-**

  Two predicates are resolved (P and Q), if either P is in the original KB or if P is an ancestor of Q in the proof tree.

- ✓ **Subsumption:-**

  A strategy which eliminate all sentences that are more specific than the existing sentence.

  For example:- if P(x) is in the KB, then there is no sense in adding P(A) to KB.

2.10.6 Theorem Provers (or) Automated reasoners

- Theorem provers use resolution or some other inference procedure to prove sentences in full first order logic, often used for mathematical and scientific reasoning tasks.
- For Example:- MRS, LIFE.

Logic Programming language (PROLOG)	**Theorem Provers**
Handles only the horn clauses	Handles full FOL representation
Choice of writing sentences in different form with same meaning affects the execution order	Does not affects the execution order derives the same conclusion
Example:- writing $A \leftarrow B \wedge C$ instead of $A \leftarrow C \wedge B$ affects the execution of the program	Example:- User can write either $A \leftarrow B \wedge C$ or $A \leftarrow C \wedge B$ or another form $\neg B \leftarrow C \wedge \neg A$ and the results will be exactly same.

Design of a Theorem Prover:-

- An example for theorem prover is: OTTER=> Organized Techniques for Theorem proving and Effective Research, with particular attention to its control strategy.
- To prepare a problem for OTTER, the user must divide the knowledge into four parts;
  1. **SetOfSupport (SOS):** A set of clauses, which defines the important facts about the problem. Resolution step resolves the member of SOS with another axiom.

2. **Usable axioms: -** A set of axioms that are outside the set of support, provides background knowledge about the problem area. The boundary between SOS and the axiom is up to the user's judgement.
3. **Rewrites (or) Demodulators :-** A set of equations evaluated or reduced from left-to-right order (i.e.) x + 0 = x in which x + 0 should be replaced by the term x.
4. A set of parameters and clauses that defines the control strategy that is,
    a. Heuristic function :- to control the search
    b. Filtering function :- eliminates some subgoals which are uninteresting

2.10.7 Execution:-

- Resolution takes place by combining the element of SOS with useful axiom, generally prefers unit strategy.
- The process continuous until a reputation is found or there are no more clauses in the SOS.
- The following shows the algorithm of execution,

Algorithm:-

```
Procedure OTTER(sos, usable)

      Inputs: sos, a set of support-clauses defining the problem (global variable)

              Usable background knowledge potentially relevant to the problem
      Repeat:
              clause        the lightest member of sos
                      ←—move clause from sos to usable
                          PROCESS (INFER(clause, usable),sos)
      Until sos = [] or a refutation has been found
function INFER(clause, usable) returns clauses
              resolve clause with each member of usable
              return the resulting clauses after applying FILTER

Procedure PROCESS (clauses, sos)
    for each clause in clauses do
        clause        SIMPLIFY (clause)
        merge ←identical literals
        discard clause if it is a tautology
        sos        [clause | sos]
        if cl←ause has no literals then a refutation has been found
        if clause has one literal then look for unit refutation
    end
```

2.10.8 Extending Prolog:-

- Theorem prover can be build using prolog compiler as a base and a sound complete reasoned of full first order logic is done using PTTP.
- Where PTTP is Prolog Technology Theorem Prover.

- PTTP includes five significant changes to prolog to restore the completeness and expensiveness.

Prolog	**PTTP**
Depth First Search	Iterative deepening search
Not possible like a PTTP implementation	Negated literals are allowed (i.e.) in the implementation P, $\neg$ P can be derived using two separate routines
Locking is not allowed	Locking is allowed (i.e.) A clause with n atoms is stored as n different rules. Example:- $A \Leftarrow B \wedge X$ is equivalent to $\neg B \Leftarrow X \wedge \neg A, \neg X \Leftarrow B \wedge \neg A$
Inference is not complete	Inference is complete since refutation is allowed
Unification is less efficient	Unification is more eficient

Drawback of PTTP:-

- Each inference rule is used by the system both in its original form and contrapositive form Example:- $(f(x,y) = f(a,b)) \Leftarrow (x = a) \wedge (y = b)$
- Prolog proves that two f terms are equal, But PTTP would also generate the contrapositive $(x \neq a) \Leftarrow (f(x,y) \neq f(a,b)) \wedge (y = b)$
- This is a wasteful way to prove that any two terms x and a are different.

****

SECOND UNIT-II LOGICAL REASONING FINISHED

****