UNIT-02

Knowledge Representation

UNIT-02/LECTURE-01

Knowledge Representation:

For the purpose of solving complex problems c\encountered in AI, we need both a large amount of knowledge and some mechanism for manipulating that knowledge to create solutions to new problems. A variety of ways of representing knowledge (facts) have been exploited in AI programs. In all variety of knowledge representations, we deal with two kinds of entities.

A. Facts: Truths in some relevant world. These are the things we want to represent.

B. Representations of facts in some chosen formalism. these are things we will

Actually be able to manipulate.

One way to think of structuring these entities is at two levels: (a) the knowledge level, at which facts are described, and (b) the symbol level, at which representations of objects at the knowledge level are defined in terms of symbols that can be manipulated by programs.

The facts and representations are linked with two-way mappings. This link is called representation mappings. The forward representation mapping maps from facts to representations. The backward representation mapping goes the other way, from representations to facts.

One common representation is natural language (particularly English) sentences. Regardless of the representation for facts we use in a program, we may also need to be concerned with an English representation of those facts in order to facilitate getting information into and out of the system. We need mapping functions from English sentences to the representation we actually use and from it back to sentences.

What to Represent?

Let us first consider what kinds of knowledge might need to be represented in AI systems:

Objects

-- Facts about objects in our world domain. e.g. Guitars have strings, trumpets are brass instruments.

Events

-- Actions that occur in our world. e.g. Steve Vai played the guitar in Frank Zappa's Band.

Performance

-- A behaviour like playing the guitar involves knowledge about how to do things.

Meta-knowledge

-- Knowledge about what we know. e.g. Bobrow's Robot who plan's a trip. It knows that it can read street signs along the way to find out where it is.

Thus in solving problems in AI we must represent knowledge and there are two entities to deal with:

Facts

-- truths about the real world and what we represent. This can be regarded as the knowledge level

Representation of the facts

Which we manipulate. This can be regarded as the symbol level since we usually define the representation in terms of symbols that can be manipulated by programs.

We can structure these entities at two levels

the knowledge level

-- at which facts are described

the symbol level

-- at which representations of objects are defined in terms of symbols that can be manipulated in programs (see Fig. 5)

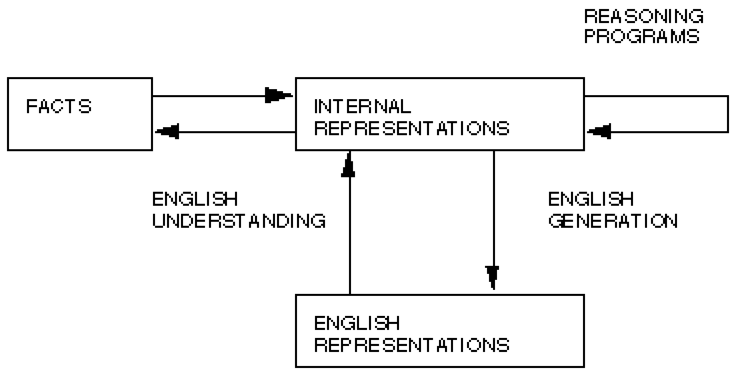


Fig 5 Two Entities in Knowledge Representation

English or natural language is an obvious way of representing and handling facts. Logic enables us to consider the following fact: spot is a dog as dog(spot) We could then infer that all dogs have tails with: tex2html_wrap_inline7154 : dog(x) tex2html_wrap_inline7156 hasatail(x) We can then deduce:

hasatail(Spot)

Using an appropriate backward mapping function the English sentence Spot has a tail can be generated.

The available functions are not always one to one but rather are many to many which is a characteristic of English representations. The sentences All dogs have tails and every dog has a tail both say that each dog has a tail but the first could say that each dog has more than one tail try substituting teeth for tails. When an AI program manipulates the internal representation of facts these new representations should also be interpretable as new representations of facts.

Consider the classic problem of the mutilated chess board. Problem In a normal chess board the opposite corner squares has been eliminated. The given task is to cover all the squares on the remaining board by dominoes so that each domino covers two squares. No overlapping of dominoes is allowed, can it be done. Consider three data structures

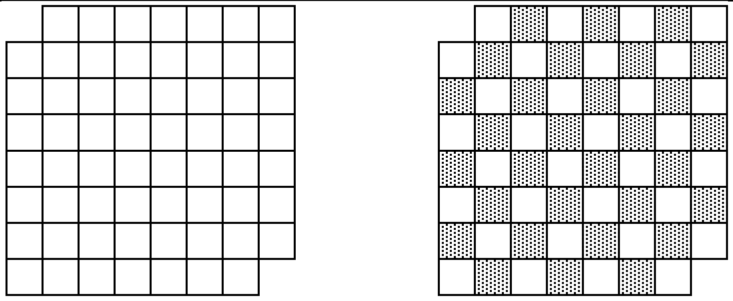


Fig. 3.1 Mutilated Checker

The first two are illustrated in the diagrams above and the third data structure is the number of black squares and the number of white squares. The first diagram loses the colour of the squares and a solution is not east to see; the second preserves the colours but produces no easier path whereas counting the number of squares of each colour giving black as 32 and the number of white as 30 yields an immediate solution of NO as a domino must be on one white square and one black square, thus the number of squares must be equal for a positive solution.

Using Knowledge

We have briefly mentioned where knowledge is used in AI systems. Let us consider a little further to what applications and how knowledge may be used.

Learning
-- acquiring knowledge. This is more than simply adding new facts to a knowledge base. New data may have to be classified prior to storage for easy retrieval, etc.. Interaction and inference with existing facts to avoid redundancy and replication in the knowledge and and also so that facts can be updated.

Retrieval
-- The representation scheme used can have a critical effect on the efficiency of the method. Humans are very good at it.

Many AI methods have tried to model human (see lecture on distributed reasoning)

Reasoning
-- Infer facts from existing data.

If a system on only knows:

Miles Davis is a Jazz Musician.

All Jazz Musicians can play their instruments well.

If things like Is Miles Davis a Jazz Musician? or Can Jazz Musicians play their instruments well? Are asked then the answer is readily obtained from the data structures and procedures.

However a question like Can Miles Davis plays his instrument well? requires reasoning.

The above are all related. For example, it is fairly obvious that learning and reasoning involve retrieval etc.

Properties for Knowledge Representation Systems

The following properties should be possessed by a knowledge representation system.

Representational Adequacy

-- the ability to represent the required knowledge;

Inferential Adequacy

- the ability to manipulate the knowledge represented to produce new knowledge corresponding to that inferred from the original;

Inferential Efficiency

- the ability to direct the inferential mechanisms into the most productive directions by storing appropriate guides;

Acquisitional Efficiency

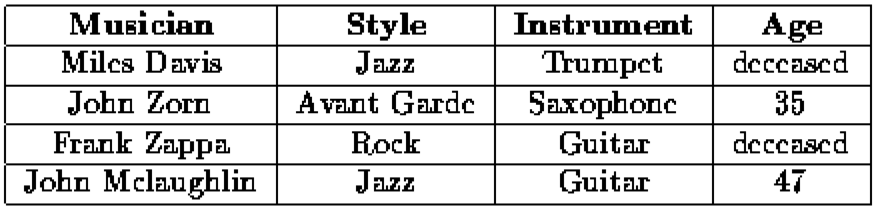
- the ability to acquire new knowledge using automatic methods wherever possible rather than reliance on human intervention.

Approaches to Knowledge Representation

Simple relational knowledge

The simplest way of storing facts is to use a relational method where each fact about a set of objects is set out systematically in columns. This representation gives little opportunity for inference, but it can be used as the knowledge basis for inference engines.

- Simple way to store facts.
- Each fact about a set of objects is set out systematically in columns (Fig. 7).
- Little opportunity for inference.
- Knowledge basis for inference engines.



Musician	Style	Instrument	Age
Miles Davis	Jazz	Trumpet	deceased
John Zorn	Avant Garde	Saxophone	35
Frank Zappa	Rock	Guitar	deceased
John Mclaughlin	Jazz	Guitar	47

Figure: Simple Relational Knowledge

We can ask things like:

Who is dead?

Who plays Jazz/Trumpet etc.?

This sort of representation is popular in database systems.

Inheritable knowledge

Relational knowledge is made up of objects consisting of

- attributes
- corresponding associated values.

We extend the base more by allowing inference mechanisms:

- Property inheritance
- Elements inherit values from being members of a class.
- Data must be organised into a hierarchy of classes (Fig. 8).

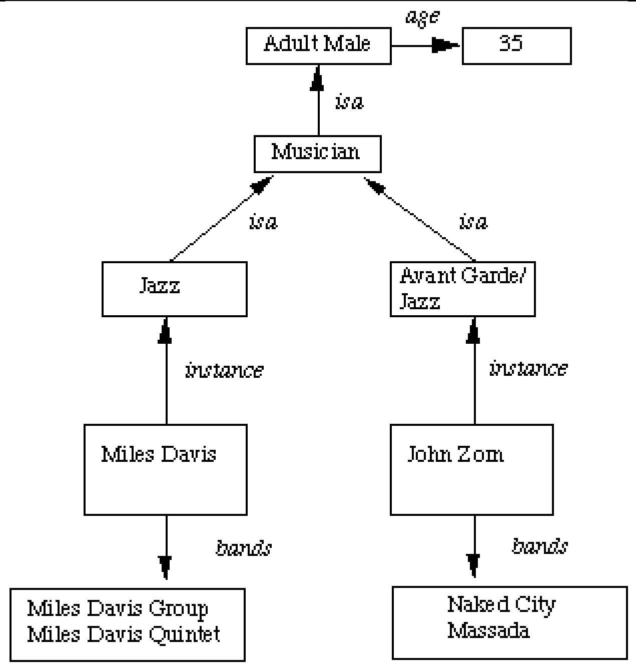


Fig. 8 Property Inheritance Hierarchy

- Boxed nodes -- objects and values of attributes of objects.
- Values can be objects with attributes and so on.
- Arrows -- point from object to its value.
- This structure is known as a slot and filler structure, semantic network or a collection of frames.

The algorithm to retrieve a value for an attribute of an instance object:

- Find the object in the knowledge base
- If there is a value for the attribute report it
- Otherwise look for a value of instance if none fail
- Otherwise go to that node and find a value for the attribute and then report it
- Otherwise search through using isa until a value is found for the attribute.

Inferential Knowledge

Represent knowledge as formal logic:

All dogs have tails $\forall x$: *dog(x)* $\rightarrow$*hasatail(x)* Advantages:

- A set of strict rules.
  - Can be used to derive more facts.
  - Truths of new statements can be verified.
  - Guaranteed correctness.
- Many inference procedures available to in implement standard rules of logic.

- Popular in AI systems. *e.g* Automated theorem proving.

Procedural Knowledge

Basic idea:

- Knowledge encoded in some procedures
- small programs that know how to do specific things, how to proceed.
- e.g a parser in a natural language understander has the knowledge that a noun phrase may contain articles, adjectives and nouns. It is represented by calls to routines that know how to process articles, adjectives and nouns.

Advantages:

- Heuristic or domain specific knowledge can be represented.
- Extended logical inferences, such as default reasoning facilitated.
- Side effects of actions may be modelled. Some rules may become false in time. Keeping track of this in large systems may be tricky.

Disadvantages:

- Completeness -- not all cases may be represented.
- Consistency -- not all deductions may be correct.

e.g If we know that Fred is a bird we might deduce that Fred can fly. Later we might discover that Fred is an emu.

- Modularity is sacrificed. Changes in knowledge base might have far-reaching effects.
- Cumbersome control information.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Explain the various approaches of knowledge representation schema with the help of examples.	June.2014	7
Q.2	Explain knowledge representation & problems	June.2013	10

	associated with it with examples		
Q.3	Discuss various approaches of knowledge representation with the help of examples	June.2011	10

UNIT-02/LECTURE-02

Problems in representing knowledge:

Below are listed issues that should be raised when using a knowledge representation technique:

Important Attributes

-- Are there any attributes that occur in many different types of problem?

There are two instance and isa and each is important because each supports property inheritance.

Relationships

-- What about the relationship between the attributes of an object, such as, inverses, existence, techniques for reasoning about values and single valued attributes. We can consider an example of an inverse in

band(John Zorn,Naked City)

This can be treated as John Zorn plays in the band Naked City or John Zorn's band is Naked City.

Another representation is band = Naked City

band-members = John Zorn, Bill Frissell, Fred Frith, Joey Barron

Granularity

-- At what level should the knowledge be represented and what are the primitives. Choosing the Granularity of Representation Primitives are fundamental concepts such as holding, seeing, playing and as English is a very rich language with over half a million words it is clear we will find difficulty in deciding upon which words to choose as our primitives in a series of situations.

If Tom feeds a dog then it could become:

feeds(tom, dog)

If Tom gives the dog a bone like:

gives(tom, dog,bone) Are these the same?

In any sense does giving an object food constitute feeding?

If give(x, food) $\rightarrow$ feed(x) then we are making progress.

But we need to add certain inferential rules.

In the famous program on relationships Louise is Bill's cousin How do we represent this? louise = daughter (brother or sister (father or mother(bill))) Suppose it is Chris then we do not know if it is Chris as a male or female and then son applies as well.

Clearly the separate levels of understanding require different levels of primitives and these need many rules to link together apparently similar primitives.
Obviously there is a potential storage problem and the underlying question must be what level of comprehension is needed.

Logic Knowledge Representation

We briefly mentioned how logic can be used to represent simple facts in the last lecture. Here we will highlight major principles involved in knowledge representation. In particular predicate logic will be met in other knowledge representation schemes and reasoning methods.

A more comprehensive treatment is given in the third year Expert Systems course. Symbols used The following standard logic symbols we use in this course are:

For all $\forall$

There exists $\exists$

Implies $\rightarrow$

Not $\neg$

Or $\vee$

And $\wedge$

Propositional Logic.

Propositional logic is a special case of FOPL. It is simple to deal with and a decision procedure exists for it. We can easily represent real world facts as logical propositions. Propositions are elementary atomic sentences, written as formula or well formed formulas (wffs). Propositions may either be true or false . some examples of simple propositions are

it is raining

snow is white

people live on moon

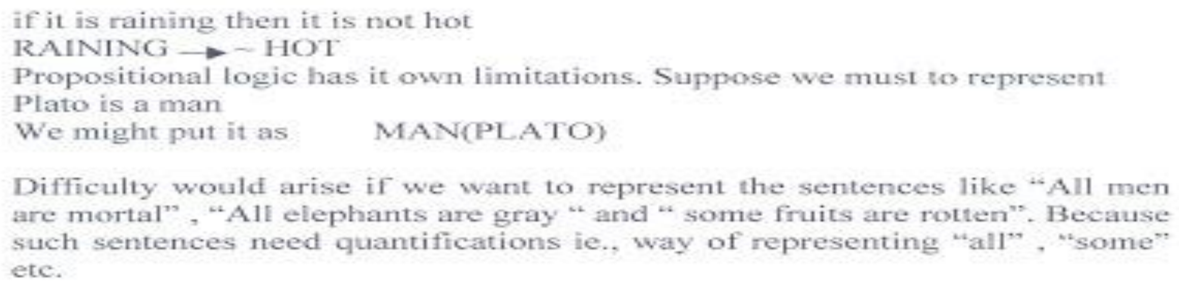
Compound Propositions are formed from wffs using logical connectives “ not, and or , if …then (implication) and if … and only if". For example :it is raining and wind is blowing" is a compound proposition.

As example of prepositional logic is given below

it is raining RAINING

it is hot HOT

it is windy WINDY



if it is raining then it is not hot
RAINING → ~ HOT
Propositional logic has it own limitations. Suppose we must to represent
Plato is a man
We might put it as MAN(PLATO)

Difficulty would arise if we want to represent the sentences like "All men are mortal" , "All elephants are gray " and " some fruits are rotten". Because such sentences need quantifications ie., way of representing "all" , "some" etc.

It is a way of representing knowledge.

In logic and mathematics, a propositional calculus or logic is a formal system in which formulae representing propositions can be formed by combining atomic propositions using logical connectives

Sentences considered in propositional logic are not arbitrary sentences but are the ones that are either true or false, but not both. This kind of sentences are called propositions.

Example

Some facts in propositional logic:

It is raining.	-	RAINING
It is sunny	-	SUNNY
It is windy	-	WINDY
If it is raining ,then it is not sunny	-	RAINING -> ¬ SUNNY

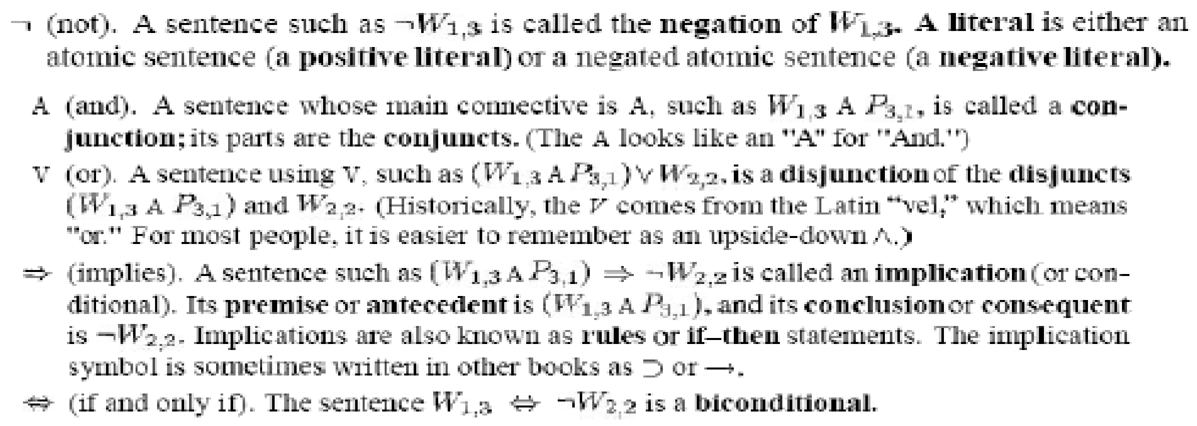
the elements of propositional logic

Simple sentences which are true or false are basic propositions. Larger and more complex sentences are constructed from basic propositions by combining them with connectives. Thus propositions and connectives are the basic elements of propositional logic. Though there are many connectives, we are going to use the following five basic connectives here:
NOT, AND, OR, IF_THEN (or IMPLY), IF_AND_ONLY_IF. They are also denoted by the symbols:
↔, respectively.

The connectives used in propositional logic.

The syntax of propositional logic defines the allowable sentences. The atomic sentences-the indivisible syntactic elements-consist of a single proposition symbol. Each such symbol stands for a proposition that can be true or false. We will use uppercase names for symbols: P, Q, R, and so on.

Complex sentences are constructed from simpler sentences using logical connectives. There are five connectives in common use:



$\neg$ (not). A sentence such as $\neg W_{1,3}$ is called the **negation of** $W_{1,3}$. **A literal** is either an atomic sentence (a **positive literal**) or a negated atomic sentence (a **negative literal**).

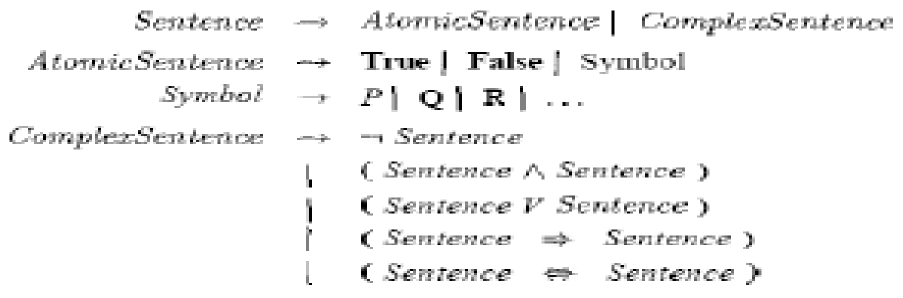
$\land$ (and). A sentence whose main connective is $\land$, such as $W_{1,3} \land P_{3,1}$, is called a **conjunction;** its parts are the **conjuncts.** (The $\land$ looks like an "A" for "And.")

$\lor$ (or). A sentence using $\lor$, such as $(W_{1,3} \land P_{3,1}) \lor W_{2,2}$, is a **disjunction** of the **disjuncts** $(W_{1,3} \land P_{3,1})$ and $W_{2,2}$. (Historically, the $\lor$ comes from the Latin "vel," which means "or." For most people, it is easier to remember as an upside-down $\land$.)

$\Rightarrow$ (implies). A sentence such as $(W_{1,3} \land P_{3,1}) \Rightarrow \neg W_{2,2}$ is called an **implication** (or conditional). Its **premise** or **antecedent** is $(W_{1,3} \land P_{3,1})$, and its **conclusion** or **consequent** is $\neg W_{2,2}$. Implications are also known as **rules** or **if–then** statements. The implication symbol is sometimes written in other books as $\supset$ or $\rightarrow$.

$\Leftrightarrow$ (if and only if). The sentence $W_{1,3} \Leftrightarrow \neg W_{2,2}$ is a **biconditional.**

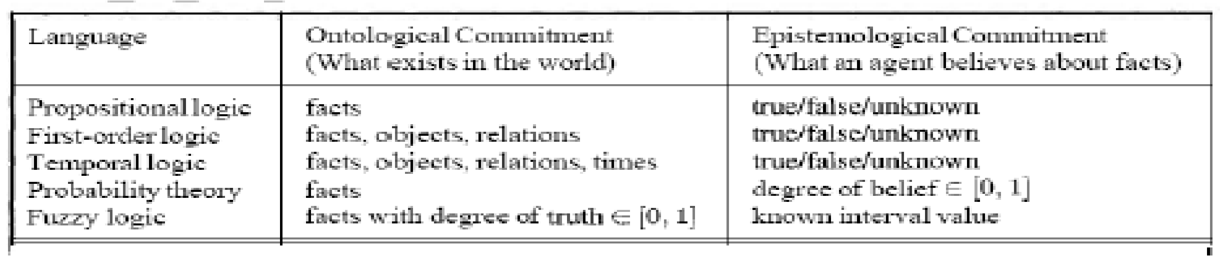
a formal grammar of propositional logic:



$$
\begin{array}{rcl}
\textit{Sentence} & \rightarrow & \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} & \rightarrow & \textbf{True} \mid \textbf{False} \mid \text{Symbol} \\
\textit{Symbol} & \rightarrow & P \mid \mathbf{Q} \mid \mathbf{R} \mid \ldots \\
\textit{ComplexSentence} & \rightarrow & \neg\ \textit{Sentence} \\
 & \mid & (\ \textit{Sentence} \land \textit{Sentence}\ ) \\
 & \mid & (\ \textit{Sentence} \lor \textit{Sentence}\ ) \\
 & \mid & (\ \textit{Sentence} \Rightarrow \textit{Sentence}\ ) \\
 & \mid & (\ \textit{Sentence} \Leftrightarrow \textit{Sentence}\ )
\end{array}
$$

A BNF (Backus–Naur Form) grammar of sentences in propositional logic.

Compare different knowledge representation languages.



Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes about facts)
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $\in [0, 1]$
Fuzzy logic	facts with degree of truth $\in [0, 1]$	known interval value

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	What do you understand by the representation of Knowledge? Explain the various problems in representing knowledge.	June.2012	10
Q.2	What is propositional logic? Explain the knowledge	June.2011	10

	representation using propositional logic.		

UNIT-02/LECTURE-03

Predicate logic

Predicate is a proposition in which some of the Boolean values are replaced by Boolean-valued functions and quantified expression. Predicate combines these functions using the operators of propositional calculus and quantifies like ∀ (for-all) and ∃ (there-exists). Some authors do call predicates as **propositional functions**.

Boolean-valued function is one with one or more variables that gives a Boolean result. For example, prime(n) is a function that returns true if **n** is prime and false if otherwise. ∀x(citizen(x, India)) returns true if everyone on the planet is a citizen of India and false otherwise. ∃x(citizen(x, India)) returns true if at least one person on the planet is a citizen of India and false otherwise.

Quantifiers are specified as (with X as variable and P as proposition):

Name	Example	Meaning
Universal	∀X.P	For all X, P is true
Existential	∃X.P	There exists a value of X such that P is true
Unique Existence	∃!X.P	There exists an unique value of X such that P is true

In can be inferred that:

1. Negation of a Universal Specification: ¬∀xQ(x) is equivalent to ∃x ¬Q(x).
2. Negation of a Existential Specification: ¬∃xQ(x) is equivalent to ∀x ¬Q(x).
3. For any predicate Q(x, y):
    a. Commutation of a Universal Quantifiers: ∀x∀y Q(x, y) is equivalent to ∀y∀x Q(x, y)
    b. Commutation of a Existential Quantifiers: ∃x∃y Q(x, y) is equivalent to ∃y∃x Q(x, y)
    c. ∃x∀y Q(x, y) → ∀y∃x Q(x, y)

In this example, all X who are colleagues are persons: ∀X.(colleague(X) ⊃ person(X))

In this example, there exists X whom the mother married is a male.

∃X.(mother(hema, X) ∩male(X))

Deduction Rules (Rules of inference): Things can be deduced from the given propositional logic using certain rules of inference and rules of replacement. Rules of inferences are generally represented in the form [Premise/Conclusion] as:

$$\frac{\text{Premise}}{\text{Conclusion}}$$

Conjunction: Given a derivation, if the first step is proposition p and the second step is proposition q, then it could be concluded as conjunction of p and q. Conjunction can also be called as conjunction introduction or &-introduction or logical multiplication.

p/q could be written as p ∩ q

Simplification: Conjunctions of p and q can be concluded as p or q. Simplification can also be called as conjunction elimination or &-elimination.

(p ∩q)/p OR (p ∩ q)/q

Addition: If the derivation is proposition p, then the conclusion might be a disjunction of p and any proposition q. Addition can also be called as disjunction introduction or ・–introduction.

p/(p ∪ q) OR q/(p ∪ q)

Disjunctive Syllogism: If the first step of derivation is proposition p ・ q and the next step is ・p, then the conclusion is q.

(p ∪q)/ ¬p/q

Modus Ponens: If the first step of derivation is proposition p ・ q and the next step is p, then the conclusion is q. Modus Ponens is also called as modus ponendo ponens or detachment or a type of ・-elimination.

(p → q)/p/q

Modus Tollens: If the first step of derivation is proposition p ・ q and the next step is ・q, then the conclusion is ・p. Modus Tollens is also called modus tollendo tollens or a type of ・-elimination.

$(p \rightarrow q)/\neg q/\neg p$

Hypothetical Syllogism: If the first step of derivation is proposition $p \rightarrow q$ and the next step is $q \rightarrow r$, then the conclusion is $p \rightarrow r$. Hypothetical syllogism is also called as chain reasoning or chain deduction.

$$(p \rightarrow q)/(q \rightarrow r)/(p \rightarrow r)$$

Constructive Dilemma: If the first step of derivation is proposition $(p \rightarrow r) \,\&\, (q \rightarrow s)$ and the next step is $(p \cup q)$, then the conclusion is $(r \cup s)$.

$$((p \rightarrow r) \,\&\, (q \rightarrow s))/(p \cup q)/(r \cup s)$$

Another example of Constructive Dilemma: $(p \cup q)/(p \rightarrow r)/(q \rightarrow s)/(r \cup s)$

Destructive Dilemma: If the first step of derivation is proposition $(\neg p \cup \neg q)$, the second step is $(r \rightarrow p)$, and the next step is $(s \rightarrow q)$, then the conclusion is $(\neg r \cup \neg s)$.

$$(\neg p \cup \neg q)/(r \rightarrow p)/(s \rightarrow q)/(\neg r \cup \neg s)$$

Absorption: If the first step of derivation is proposition $(p \rightarrow q)$, then the conclusion is $(p \rightarrow (p \,\&\, q))$.

$$(p \rightarrow q)/(p \rightarrow (p \,\&\, q))$$

Excluded Middle Introduction: During any step of the derivation, we could include a disjunction of a proposition p, along with the negation of it. This inclusion will have no change on the derivation. This could be represented as $(p \cup \neg p)$.

Universal Specification: Given an assertion of the form $\forall x P(x)$, we could conclude $P(s)$, where s is any value of the variable x. $\forall x P(x)/P(s)$

Existential Specification: Given an assertion of the form $\exists x P(x)$, we could conclude P(s), where s is a constant and not a variable. $\exists x P(x)/P(s)$

Universal Generalization: Given a derivation of the form Q(x) where x is a free variable which could be represented by any value within a specific domain, then we could conclude the derivation as $\forall x P(x)$.

Existential Generalization: Given a derivation of the form Q(x) where x is a constant value, then we could conclude the derivation as $\exists x P(x)$.

Deduction Rules (Rules of Replacement): In any given derivation, there are ways by which we could replace a given proposition with its equivalent. These substitutions are based on the rules of replacement. Rules of replacement are different from rules of inference because here we don't infer anything but instead just provide an equivalent using different symbols. Rules of inferences can be applied to the derivation as a whole only but rules of replacement can be applied on parts of the derivation. Rules of inferences are unidirectional but rules of replacement are bidirectional.

The following table summarizes various properties that could be applied on predicate calculus. The specific property of deMorgan's property helps us to remove disjunctions or conjunctions without changing the actual meaning.

Property	Meaning	Meaning
Double Negation ($\neg$-elimination)	$\neg\neg p \equiv p$	
Commutativity	$p \vee q \equiv q \vee p$	$p \wedge q \equiv q \wedge p$
Associativity	$(p \vee q) \vee r \equiv p \vee (q \vee r)$	$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
Tautology	$p \equiv p \vee p$	$p \equiv p \wedge p$
Distributivity	$p \vee q \wedge r \equiv (p \vee q) \wedge (p \vee r)$	$p \wedge (q \vee r) \equiv p \wedge q \vee p \wedge r$
Transposition (Contraposition)	$p \supset q \equiv \neg q \supset \neg p$	
Exportation	$p \supset (q \supset r) \equiv (p \wedge q) \supset r$	
Idempotence	$p \vee p \equiv p$	$p \wedge p \equiv p$
Identity	$p \vee \neg p \equiv$ true	$p \wedge \neg p \equiv$ false

deMorgan	$\neg(p \vee q) \equiv \neg p \wedge \neg q$	$\neg(p \wedge q) \equiv \neg p \vee \neg q$
Implication	$p \supset q \equiv \neg p \vee q$	
Quantification	$\neg\forall x P(x) \equiv \exists x \neg P(x)$	$\neg\exists x P(x) \equiv \forall x \neg P(x)$
Material Equivalence	$p \Leftrightarrow q \equiv (p \supset q) \wedge (q \supset p)$	$p \Leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$ $p \Leftrightarrow q \equiv (\neg p \vee q) \wedge (\neg q \vee p)$

Material equivalence can also be called as "biconditional introduction/elimination" or "▯-introduction/elimination".

Conditional and Indirect Proof: Given any proposition, the deduction could be made directly using the rules of inference or rules of replacement or just as a premise. If the deduction is straightforward, then the deduction is called as **direct deduction**.

Conditional Proof: Given the Premise and Conclusion, we wish to prove the conclusion. Most of the conditionals are of the form $p \rightarrow q$, where q is the conclusion that follows as an implication of the premise p. To prove that the conditional is true, we must prove that q follows p. Let's check whether the conclusion follows the premise, using some examples:

1. Given that $(p \rightarrow q) \wedge (q \rightarrow r)$ is the premise and $(p \rightarrow r)$ is the conclusion.

   The premise is generally represented with a right arrow preceding it.

   $|\rightarrow (p \rightarrow q) \wedge (q \rightarrow r)$ … (1)

   Given the premise, using Simplification, we could select

   $p \rightarrow q$ … (2)

   Given $p \rightarrow q$, the premise here should be p and thus, using Modus Ponens, we could conclude on q. …(3)

   From the equation (1), we could also use Simplification and select $(q \rightarrow r)$.

   From $(q \rightarrow r)$ and using (3) as the premise, we could apply Modus Ponens and infer at r.

   … (4)

   Thus, we could infer that $p \rightarrow r$ as the conclusion.

2. Let's assume that we are given that the statements are of the order: $a \rightarrow (b \vee c)$ and $a \rightarrow \neg d$ and $d \Leftrightarrow b$. we need to conclude that $a \rightarrow c$.

The premise is generally represented with a right arrow preceding it.

$|\rightarrow a \rightarrow (b \vee c)$ ... (1)

Given the implication, a is the initial premise. ... (2)

Using Material Equivalence, we could infer that:

$d \Leftrightarrow b \equiv (d \rightarrow b) \wedge (b \rightarrow d)$ ... (3)

Using Simplification, we could infer $(b \rightarrow d)$ from (3). ... (4)

Using the initial premise **a** and $(\mathbf{a} \rightarrow \neg\mathbf{d})$, we could using Simplification to attain $\neg d$.

... (5)

Using (4) and (5) and applying Modus Tollens, we could attain $\neg b$. ... (6)

Using the initial premise **a** and $(\mathbf{a} \rightarrow (\mathbf{b} \vee \mathbf{c}))$, we could use Modus Ponens to attain $(b \vee c)$. ... (7)

Using (6) and (7) and applying Disjunctive Syllogism, we could infer to **c**.

Thus, we could infer $\mathbf{a} \rightarrow \mathbf{c}$, which implies **c** (conclusion) follows **a** (premise).

Indirect Proof: Indirect Proof is also called as proof by reduction ad absurdum (reduce to the absurd). In indirect proof, we have to conclude that the basis of the premise is false. Initially, we assume that the premise is true, which is the opposite of what we have to conclude. Then we reach a contradiction having the conjunction of the premise p (or any other intermediate sentence q) and negation of the premise $\neg p$ (or any other intermediate sentence �q).

Let's consider that the premise is **p** and an intermediate step contains $\mathbf{q} \wedge \neg\mathbf{q}$. As the intermediate step is achievable from the premise, using conditional proof, we could represent $\mathbf{p} \rightarrow (\mathbf{q} \wedge \neg\mathbf{q})$.

Using Transposition $[p \supset q \equiv \neg q \supset \neg p]$, we can attain: $\neg(\mathbf{q} \wedge \neg\mathbf{q}) \rightarrow \neg\mathbf{p}$

Using DeMorgan's Law $[\neg(p \wedge q) \equiv \neg p \vee \neg q]$, we can attain: $(\neg\mathbf{q} \vee \mathbf{q}) \rightarrow \neg\mathbf{p}$

... (1)

$(\neg\mathbf{q} \vee \mathbf{q})$ is an excluded middle introduction. ... (2)

Using (1) and (2) and applying Modus Ponens, we could conclude $\neg\mathbf{p}$ which is the opposite of the premise p. As the negation of the premise is true, then the premise is concluded to be false.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Convert the following sentences to predicate logic: i) All Romans were either loyal to Caesar or hated him. ii) Everyone is loyal to someone iii) Marcus tried to assassinate Caesar. iv) Caesar was ruler	June.2012	10
Q.2	Convert the following sentences to predicate logic: i) Marcus was a man. ii) All men are mortal. iii) No mortal lives longer than 150 years.	June.2011	10
Q.3	Convert the following sentences to predicate logic: i) Marcus was a man. ii) All pomperians were romans. iii) No mortal lives longer than 150 years. iv) 4 is an even number v) All even numbers are divisible by 2.	June.2013	10

http://iiscs.wssu.edu/drupal/node/3661

UNIT-02/LECTURE-04

Clausal Form: Clausal form is one in which there is the **antecedent** and the **consequent**. Consequent depends upon the antecedent. Antecedent needs to be verified before the consequent can be concluded. Existential quantifiers are not required and universal quantifiers are implicit in the use of variables in atomic proposition. No operators other than conjunction [$\cap$] and disjunction [$\cup$] are needed. We can represent the clausal form in any one of the following ways:

$b_1 \cup b_2 \cup \cdots \cup b_n \subset a_1 \cap a_2 \cap \cdots \cap a_m$

$b_1 \cup b_2 \cup \cdots \cup b_n \leftarrow a_1 \cap a_2 \cap \cdots \cap a_m$

$a_1 \cap a_2 \cap \cdots \cap a_m \rightarrow b_1 \cup b_2 \cup \cdots \cup b_n$

means if all the A's are true, then at least one B is true. Here, it is obvious that all A's indicate every A and thus it is universally quantified. If m = 0, then there is no antecedent to be tested and thus the consequent is implied and called as a **fact**.

plays(saleem, cricket) $\subset$ plays(saleem, game) $\cap$ game(cricket)

In this example, if saleem plays a game and the game is cricket then it implies that saleem plays cricket. Here, the person name (saleem) and the game name (cricket) are fixed and thus existential quantifiers are not needed.

Resolution and Unification: We can use propositions to discover new theorems from known axioms or assertions and other known theorems. Resolution is a principle of inference that allows inferred propositions to be computed from given propositions. As an example, if $a \subset b$ and $c \subset d$ and suppose, **a** is identical to **d**, then the Resolution is $c \subset b$. Resolution is also called as the **control strategy** of the logical programming language.

Few Examples:

older(a,b) $\subset$ mother(a,b)

wiser(a,b) $\subset$ older(a,b)

Resolution: wiser(a,b) $\subset$ mother(a,b)

By stating that mothers are older and older people are wiser, we resolve that mothers are wiser.

The most common form of resolution is called as **binary resolution**. In binary resolution, one clause contains a literal and the other contains the negation of the literal. In this case, a new clause is produced by removing the literal and its negation and then making the disjunction of the remaining elements of both the clauses.

Unit Resolution	$\dfrac{P(x) \vee Q(x),\ \neg P(A)}{Q(A)}$
Resolution	$\dfrac{P(x) \vee Q(x),\ \neg Q(x) \vee R(x)}{P(x) \vee R(x)}$

Resolution requires the given logical statements to be converted into **Conjunctive Normal Form** (CNF) or clause form. A formula is said to be in CNF form if it is made up of conjunction of clauses, where a clause is made up of disjunction of literals. This is required for resolution because resolution acts on a pair of disjunctions to produce a new disjunction. The use of conjunction of clauses is required because of the assumption that all clauses are assumed to be true at the same time. In resolution, $P(x) \vee Q(x), \neg Q(x) \vee R(x)$ can be converted to $P(x) \vee R(x)$ because among $Q(x)$ and $\neg Q(x)$, one of them should always be true and the other should always be false. Thus $Q(x)$ and $\neg Q(x)$ is a tautology.

Clauses

A Clause is a disjunction of literals. A Ground Clause is a variable-free clause. A Horn Clause is a clause with at most one positive literal. A Definite Clause is a Horn Clause with exactly one positive Literal.

Notice that implications are equivalent to Horn or Definite clauses: (A IMPLIES B) is equivalent to ((NOT A) OR B)

(A AND B IMPLIES FALSE) is equivalent to ((NOT A) OR (NOT B)).

Formulae

A Formula is either:

- an atomic formula, or
- a Negation, i.e. the NOT of a formula, or

- a Conjunctive Formula, i.e. the AND of formulae, or
- a Disjunctive Formula, i.e. the OR of formulae, or
- an Implication, that is a formula of the form (formula1 IMPLIES formula2), or
- an Equivalence, that is a formula of the form (formula1 IFF formula2), or
- a Universally Quantified Formula, that is a formula of the form (ALL variable formula).

We say that occurrences of variable are bound in formula [we should be more precise]. Or

- a Existentially Quantified Formula, that is a formula of the form (EXISTS variable formula). We say that occurrences of variable are bound in formula [we should be more precise].

An occurrence of a variable in a formula that is not bound, is said to be free. A formula where all occurrences of variables are bound is called a closed formula, one where all variables are free is called an open formula.

A formula that is the disjunction of clauses is said to be in Clausal Form. We shall see that there is a sense in which every formula is equivalent to a clausal form.

Often it is convenient to refer to terms and formulae with a single name. Form or Expression is used to this end.

Substitutions

- Given a term s, the result [**substitution instance**] of **substituting a term t in s for a variable x**, s[t/x], is:
  - t, if s is the variable x
  - y, if s is the variable y different from x
  - F(s1[t/x] s2[t/x] .. sn[t/x]), if s is F(s1 s2 .. sn).
- Given a formula A, the result (**substitution instance**) of **substituting a term t in A for a variable x**, A[t/x], is:
  - FALSE, if A is FALSE,
  - P(t1[t/x] t2[t/x] .. tn[t/x]), if A is P(t1 t2 .. tn),
  - (B[t/x] AND C[t/x]) if A is (B AND C), and similarly for the other connectives,
  - (ALL x B) if A is (ALL x B), (similarly for EXISTS),
  - (ALL y B[t/x]), if A is (ALL y B) and y is different from x (similarly for EXISTS).

The substitution [t/x] can be seen as a map from terms to terms and from formulae to formulae. We can define similarly [t1/x1 t2/x2 .. tn/xn], where t1 t2 .. tn are terms and x1 x2 .. xn are variables, as a map, the **[simultaneous] substitution of x1 by t1, x2 by t2, .., of xn by tn**. [If all the terms t1 .. tn are variables, the substitution is called an **alphabetic variant**, and if they are ground terms, it is called a **ground substitution**.] Note that a simultaneous substitution is not the same as a sequential substitution.

Unification

Given two substitutions S = [t1/x1 .. tn/xn] and V = [u1/y1 .. um/ym], the **composition** of S and V, S . V, is the substitution obtained by:
o Applying V to t1 .. tn [the operation on substitutions with just this property is called **concatenation**], and
o adding any pair uj/yj such that yj is not in {x1 .. xn}.

For example: [G(x y)/z].[A/x B/y C/w D/z] is [G(A B)/z A/x B/y C/w].

Composition is an operation that is associative and non commutative

A set of forms f1 .. fn is **unifiable** iff there is a substitution S such that f1.S = f2.S = .. = fn.S. We then say that S is a **unifier** of the set.

For example {P(x F(y) B) P(x F(B) B)} is unified by [A/x B/y] and also unified by [B/y].

A **Most General Unifier** (MGU) of a set of forms f1 .. fn is a substitution S that unifies this set and such that for any other substitution T that unifies the set there is a substitution V such that S.V = T. The result of applying the MGU to the forms is called a **Most General Instance** (MGI). Here are some examples:

FORMULAE	MGU	MGI
--(P x), (P A)	[A/x]	(P A)
--(P (F x) y (G y)), (P (F x) z (G x))	[x/y x/z]	(P (F x) x (G x))
--(F x (G y)), (F (G u) (G z))	[(G u)/x y/z]	(F (G u) (G y))
--(F x (G y)), (F (G u) (H z))	Not Unifiable	
--(F x (G x) x), (F (G u) (G (G z)) z)	Not Unifiable	

This last example is interesting: we first find that (G u) should replace x, then that (G z) should replace x; finally that x and z are equivalent. So we need x->(G z) and x->z to be both true. This would be possible only if z and (G z) were equivalent. That cannot happen for a finite term. To recognize cases such as this that do not allow unification [we cannot replace z by (G z) since z occurs in (G z)], we need what is called an Occur Test . Most Prolog implementation use Unification extensively but do not do the occur test for efficiency reasons.

The determination of Most General Unifier is done by the **Unification Algorithm**. Here is the pseudo code for it:

FUNCTION Unify WITH PARAMETERS form1, form2, and assign RETURNS MGU, where form1 and form2 are the forms that we want to unify, and assign is initially nil.

1. Use the Find-Difference function described below to determine the first elements where form1 and form2 differ and one of the elements is a variable. Call difference-set the value returned by Find-Difference. This value will be either the atom Fail, if the two forms cannot be unified; or null, if the two forms are identical; or a pair of the form (Variable Expression).

2. If Find-Difference returned the atom Fail, Unify also returns Fail and we cannot unify the two forms.

3. If Find-Difference returned nil, then Unify will return assign as MGU.

4. Otherwise, we replace each occurrence of Variable by Expression in form1 and form2; we compose the given assignment assign with the assignment that maps Variable into Expression, and we repeat the process for the new form1, form2, and assign.

FUNCTION Find-Difference WITH PARAMETERS form1 and form2 RETURNS pair, where form1 and form2 are e-expressions.

1. If form1 and form2 are the same variable, return nil.

2. Otherwise, if either form1 or form2 is a variable, and it does not appear anywhere in the other form, then return the pair (Variable Other-Form), otherwise return Fail.

3. Otherwise, if either form1 or form2 is an atom then if they are the same atom then return nil otherwise return Fail.

4. Otherwise both form1 and form2 are lists.

Apply the Find-Difference function to corresponding elements of the two lists until either a call returns a non-null value or the two lists are simultaneously exhausted, or some elements are left over in one list.

In the first case, that non-null value is returned; in the second, nil is returned; in the third, Fail is returned

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Consider the following facts: i) The members of the friends club are suresh, megha, niket and shruti. ii) Suresh is married to megha. iii) Niket is shruti's brother. iv) The spouse of every married person in the club is also in the club. v) The last meeting of the club was at suresh's house. a) Represent these facts in predicate logic. b) Convert it into clause form. c) From the fact given above, most people would be able to decide on the truth of the following additional statements: vi) The last meeting of the club was at megha's house vii) Shruti is not married. Can you construct resolution proofs to demonstrate the	June,2006	20

	truth of each of these statements given the facts listed above? Do so if possible . otherwise give the reason.		
Q.2	Explain Unification algorithm.	June,2006	10
Q.3	Convert the following to clause form ∀X∃yP(x,y)->∃ Q(y, x)	June,2013	10

UNIT-02/LECTURE-05

Resolution Refutation: Resolution refutation is used to prove a theorem by negating the goal that should be proved. This negated goal should be added to the actual set of true axioms that are present in the given statements. After adding the negated goal, the rules of inferences are used to resolve that this logic leads to a contradiction. If the negated goal contradicts the given set of axioms, then the original goal is said to be consistent with the set of axioms.

Steps involved in resolution refutation proofs are:

1. Convert the given premises or axioms into clause form.
2. Add the negation of the goal to the set of given axioms.
3. Resolve them and produce a contradiction. Contradiction could be shown by ending up with an empty clause. Whenever there is a contradiction in the clauses, an empty clause can always be generated and thus resolution is said to be **refutation complete**.

Let's consider a simple example.

Given axioms: (1) Mango is a Tree. (2) All Trees are Plants. (3) All Trees need water

Goal: Mango Tree needs water.

1. Convert the premises into predicates.
   a. tree(Mango)
   b. ∀x (tree(x) → plants(x))
   c. ∀x (plants(x) → water(x))
2. The goal required is:
   water(mango)
3. Using Modus Ponens on Step 1 (a and b), we get
   plants(mango)
4. Using Modus Ponens on Step 2 and 1 (c), we get
   water(mango)
5. Now, convert the predicates into clause form

Predicate	**Clause Form**
tree(mango)	tree(mango)
∀x (tree(x) → plants(x))	¬tree(x) ∨ plants(x)
∀x (plants(x) → water(x))	¬plants(x) ∨ water(x)
Negate the Conclusion: ▯water(mango)	▯water(mango)

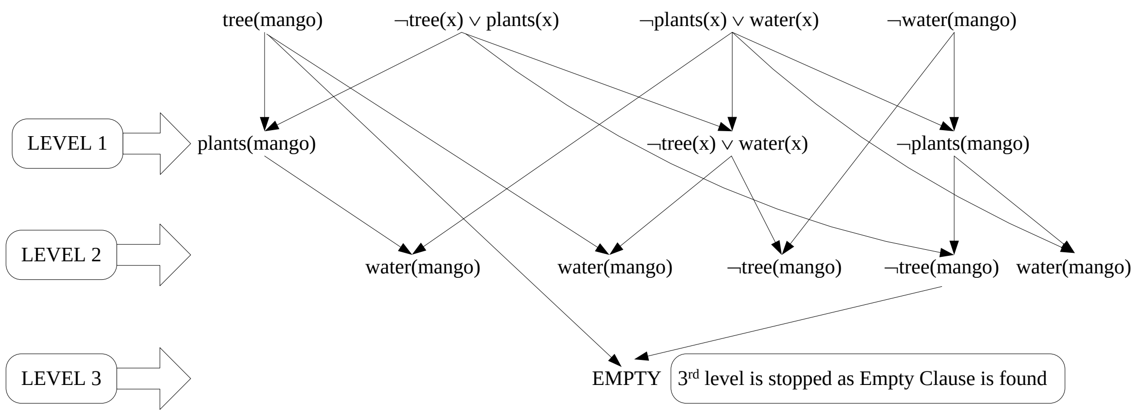
6. Combining (¬**tree**(**x**) ∨ **plants**(**x**)) and (¬**plants**(**x**) ∨ **water**(**x**)), and using resolution, we get (¬tree(x) ∨ water(x))
7. Combining **tree**(**mango**) and (¬**tree**(**x**) ∨ **water**(**x))**, we get
   water(mango)
8. Considering water(mango) and ¬water(mango), we get an empty clause.

Strategies for Resolution: Strategy for resolution defines the way by which various clauses are considered for resolution. There are many strategies present for resolution process. A strategy is called as a **complete strategy** if by using the refutation-complete rule we are guaranteed to find a refutation provided the set of clauses given is unsatisfiable. Refutation-complete rule means that we could attain unsatisfiability by using the inference rules alone, provided the set of clauses given is unsatisfiable. A set of clauses is said to be unsatisfiable if there is no interpretation available that could make the set satisfiable. Any clause (C1) is called as an ancestor of another clause (C2) only if C2 is a resolvent (result of resolution) of C1 and a different clause or C2 is a resolvent of a descendant of C1 and a different clause.

- **Breadth-First Strategy:** Breadth-First strategy is of complete strategy type. Here, every clause is compared for resolution with every other clause provided, to produce the first level of clauses. The clauses in the first level are then compared for resolution with the original clause, to produce the second level of clauses. Thus, the clauses at the nth level are generated by resolving the (n-1)th level clause elements with the original clause as well as all the previous level of clauses. Breadth-First strategy guarantees a shortest solution path as every level is exhausted before moving to the next level. If the process is continued for many levels, it is guaranteed to find a refutation if one exists and thus this strategy is said to be complete. Breadth-First strategy is better for a small set of clauses.

  Given the clauses as: tree(mango), ¬tree(x) ∨ plants(x), ¬plants(x) ∨ water(x)

  Negation of the goal: ¬water(mango)



- **Unit Preference Strategy:** Here, the resultant clause produced is preferred to have fewer literals than the parent clause. Also, it is preferred to do resolution with unit clauses. Thus, unit clauses are used for resolution whenever they are available and this causes the resolution refutation to be restricted.

  Given the clauses as: tree(mango), ¬tree(x) ∨ plants(x), ¬plants(x) ∨ water(x)

  Negation of the goal: ¬water(mango)

$$\begin{tikzcd}
\text{tree(mango)} \arrow[dr, "\{mango/x\}"] & & \neg\text{tree}(x) \vee \text{plants}(x) \arrow[dl] & \\
& \text{plants(mango)} \arrow[dr, "\{mango/x\}"] & & \neg\text{plants}(x) \vee \text{water}(x) \arrow[dl] \\
\neg\text{water(mango)} \arrow[dr] & & \text{water(mango)} \arrow[dl] & \\
& \text{EMPTY} & &
\end{tikzcd}$$

- **Set of Support Strategy:** This strategy is said to be good for large set of clauses. Given a set of input clauses (S), we can select a subset (T) of (S), which is called as the set of support. The given set of clauses is divided into two partitions: (1) set of support and (2) auxiliary set.

Make sure that the auxiliary set is not contradictory. Generally, the given set of clauses is not contradictory and the contradiction arises only with the addition of the negation of the goal. So, auxiliary set is generally made up of all the given axioms excluding the negation of the goal.

The set of support must be consisted of clauses that form the negated theorem or descendants of the negated theorem. The set of support is generally started just with the negation of the goal.

Using auxiliary set alone is not possible to derive contradictions, and thus one should take at least one clause from the set of support. The strategy requires that one of the resolvents in each resolution must be from the set of support, i.e., either should be coming from the negation of the goal or have an ancestor in the set of support.

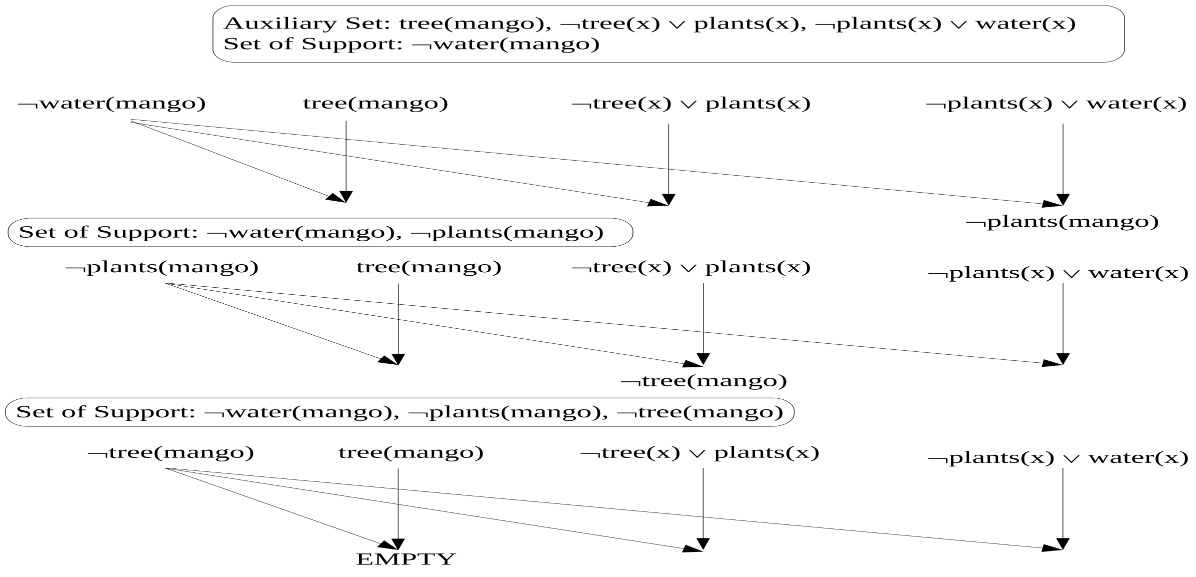
Given the clauses as: tree(mango), ¬tree(x) ∨ plants(x), ¬plants(x) ∨ water(x)

Negation of the goal: ¬water(mango)

Set of support = Negation of the goal = ¬water(mango)

All the other clauses are considered to be in the auxiliary set.

During the first iteration, ¬plants(mango) is got as the resolvent. This resolvent is not coming from the negation of the goal, but one of its ancestor is from the set of support. Its ancestors are ¬water(mango) and ¬plants(x) ∨ water(x). Now, we add the resolvent to the set of support and thus the set of support becomes ¬water(mango), ¬plants(mango). Now, we apply the new member of the set of support with the original auxiliary set. The new resolvent is added to the set of support and the process continues until an empty resolvent is obtained.



- **Linear Input Form Strategy:** Here the negated goal is taken first and resolved with any one of the original axioms but not the goal. The resultant clause is then resolved with any one of the axioms to get another new clause, which is then resolved with any one of the axioms. This process continues until an empty clause is obtained.

  Given the clauses as: tree(mango), ¬tree(x) ∨ plants(x), ¬plants(x) ∨ water(x)

  Negation of the goal: ¬water(mango)

$$\begin{tikzcd}
\neg water(mango) \arrow[dr, "\{mango/x\}"'] & & \neg plants(x) \lor water(x) \arrow[dl] & \\
& \neg plants(mango) \arrow[dr, "\{mango/x\}"'] & & \neg tree(x) \lor plants(x) \arrow[dl] \\
tree(mango) \arrow[dr] & & \neg tree(mango) \arrow[dl] & \\
& \text{EMPTY} & &
\end{tikzcd}$$

It is also noted that Linear Input Strategy allows those resolutions where one of the clauses is a member of the original set of axioms and/or the negated theorem. Linear Input Strategy is not refutation complete.

- **Ancestry Filtering Strategy:** The condition statement in this strategy is that at least one member of the clauses that are being resolved either is a member of the original set of clauses or is an ancestor of the other clause being resolved.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Write short note:- i) Resolution ii) Refutation iii) Inferencing	June.2014	10

UNIT-02/LECTURE-06

RESOLUTION IN PREDICATE LOGIC

Two literals are contradictory if one can be unified with the negation of the other. For example man(x) and man (Himalayas) are contradictory since man(x) and man(Himalayas) can be unified. In predicate logic unification algorithm is used to locate pairs of literals that cancel out. It is important that if two instances of the same variable occur, then they must be given identical substitutions. The resolution algorithm for predicate logic as follows

Let f be a set of given statements and S is a statement to be proved.

1. Covert all the statements of F to clause form.

2. Negate S and convert the result to clause form. Add it to the set of clauses obtained in 1.

3. Repeat until either a contradiction is found or no progress can be made or a predetermined amount of effort has been expended.

a) Select two clauses. Call them parent clauses.

b) Resolve them together. The resolvent will be the disjunction of all of these literals of both clauses. If there is a pair of literals T1 and T2 such that one parent clause contains Ti and the other contains T2 and if T1 and T2 are unifiable, then neither t1 nor T2 should appear in the resolvent. Here Ti and T2 are called complimentary literals.

© If the resolvent is the empty clause , then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.

Using resolution to produce proof is illustrated in the following statements.

1. Marcus was a man.

2. Marcus was a Pompeian

3. All pompeians were Roman

4. Caesar was a ruler

5. All Romans were either loyal to Caesar or hated him.

6. Everyone is loyal to someone.

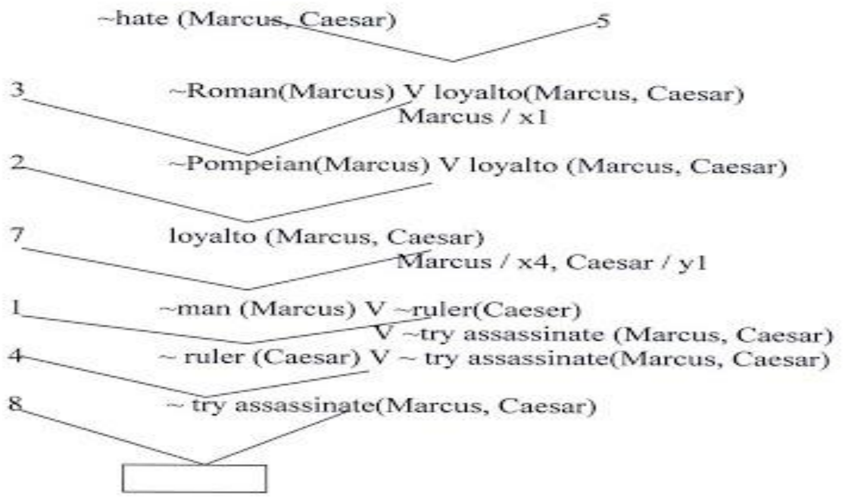
7. People only try to assassinate rulers they are not loyal to.

8. Marcus tried to assassinate Caesar.

In order to use the above resolution roofs, they must be converted to clause form as given below:

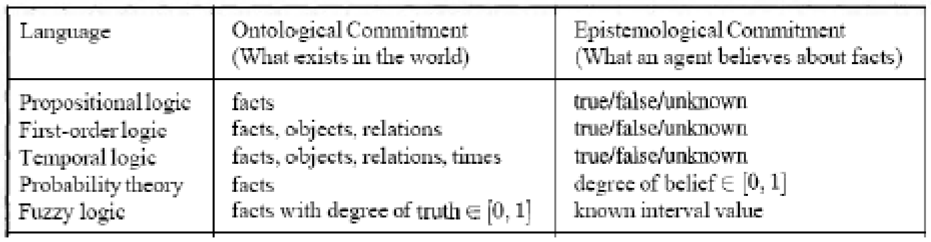
1. man (Marcus)
2. Pompeian(Marcus)
3. ~ Pompeian(x1) V roman (x1)
   (from Pompeian (x1) ---> Roman (x1))
4. ruler (caesar)
5. ~Roman (x2) V loyalto (x2, Caesar) V hate (x2, Caesar)
6. loyalto (x3, f1(x3))
7. ~man(x4) V ~ ruler(y1) V ~ try assassinate(x4,y1) ~ loyalto (x4, y1))
8. try assassinate (Marcus, Caesar)

Suppose our goal is to answer the question of the assertion S, ie., hate (Marcus, Caesar) . The resolution procedure has been illustrated below.


~hate (Marcus, Caesar)
5
3
~Roman(Marcus) V loyalto(Marcus, Caesar)
Marcus / x1
2
~Pompeian(Marcus) V loyalto (Marcus, Caesar)
7
loyalto (Marcus, Caesar)
Marcus / x4, Caesar / y1
1
~man (Marcus) V ~ruler(Caeser)
V ~try assassinate (Marcus, Caesar)
4
~ ruler (Caesar) V ~ try assassinate(Marcus, Caesar)
8
~ try assassinate(Marcus, Caesar)

The empty clause shows that ~hate (Marcus, Caesar) produces a contradiction or hate(Marcus, Caesar) will not produce a contradiction with the known statements.

Compare different knowledge representation languages



Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes about facts)
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $\in [0, 1]$
Fuzzy logic	facts with degree of truth $\in [0, 1]$	known interval value

Resolution Refutation Proofs

The basic steps

Convert the set of rules and facts into clause form (conjunction of clauses)

Insert the negation of the goal as another clause

Use resolution to deduce a refutation

If a refutation is obtained, then the goal can be deduced from the set of facts and rules.

Conversion to Normal Form

・A formula is said to be in clause form if it is of the form:

$\forall x1\ \forall x2\ \ldots\ \forall x_n\ [C_1 \wedge C_2 \wedge \cdot \cdots \wedge C_k]$

・All first-order logic formulas can be converted to clause form

・We shall demonstrate the conversion on the formula:

$\forall x\ \{p(x) \Rightarrow \cdot\ \exists z\ \{\neg \forall y\ [q(x,y) \Rightarrow p(f(x_1))] \wedge \forall y\ [q(x,y) \Rightarrow p(x)]\ \}\}$

Step1: Take the existential closure and eliminate redundant quantifiers. This introduces $\exists x1$ and eliminates $\exists z$, so:

$\forall x\ \{p(x) \Rightarrow \exists z\ \{\neg \forall y\ [q(x,y) \Rightarrow p(f(x_1))] \wedge \forall y\ [q(x,y) \Rightarrow p(x)]\ \}\}$

$\exists x_1\ \forall x\ \{p(x) \Rightarrow \{\neg \forall y\ [q(x,y) \Rightarrow p(f(x_1))] \wedge \forall y\ [q(x,y) \Rightarrow p(x)]\ \}\}$

Step 2: Rename any variable that is quantified more than once. y has been quantified twice, so:

$\exists x_1\ \forall x\ \{p(x) \Rightarrow \{\neg \forall y\ [q(x,y) \Rightarrow p(f(x_1))] \wedge \forall y\ [q(x,y) \Rightarrow p(x)]\ \}\}$

$\exists x_1\ \forall x\ \{p(x) \Rightarrow \{\neg \forall y\ [q(x,y) \Rightarrow p(f(x_1))] \wedge \forall z\ [q(x,z) \Rightarrow p(x)]\ \}\}$

Step 3: Eliminate implication.

$\exists x_1\ \forall x\ \{p(x) \Rightarrow \{\neg \forall y\ [q(x,y) \Rightarrow p(f(x_1))] \wedge \forall z\ [q(x,z) \Rightarrow p(x)]\ \}\}$

$\exists x_1\ \forall x\ \{\neg p(x) \vee \{\neg \forall y\ [\neg q(x,y) \vee p(f(x\ ))] \wedge \forall z\ [\neg q(x,z) \vee p(x)]\ \}\}$

Step 4: Move $\neg$ all the way inwards.

$\exists x_1 \forall x \{\neg p(x) \vee \{\neg \forall y [\neg q(x,y) \vee p(f(x_1))] \wedge \forall z [\neg q(x,z) \vee p(x)] \}\}$

$\exists x_1 \forall x \{\neg p(x) \vee \{ \exists y [q(x,y) \wedge \neg p(f(x_1))] \wedge \forall z [\neg q(x,z) \vee p(x)] \}\}$

Step 5: Push the quantifiers to the right

$\exists x_1 \forall x \{\neg p(x) \vee \{ \exists y [q(x,y) \wedge \neg p(f(x_1)) \wedge \forall z [\neg q(x,z) \vee p(x)] \}\}$

$\exists x_1 \forall x \{\neg p(x) \vee \{[\exists y\, q(x,y) \wedge \neg p(f(x_1)) \wedge [\forall z\, \neg q(x,z) \vee p(x)] \}\}$

Step 6: Eliminate existential quantifiers (Skolemization).

Pick out the leftmost $\exists y\, B(y)$ and replace it by $B(f(x_{i1}, x_{i2},\ldots, x_{in}))$, where:

a)x_{i1} , x_{i2} ,…, x_{in} are all the distinct free variables of $\exists y\, B(y)$ that are universally quantified to the left of $\exists y\, B(y)$, and

b)F is any n-ary function constant which does not occur already

Skolemization:

$\exists x1 \forall x \{\neg p(x) \vee$ ・$\{[\exists y\, q(x,y) \wedge \neg p(f(x1))] \wedge [\forall z\, \neg q(x,z) \vee p(x)] \}\}$

$\forall x \{\neg p(x) \vee$ ・$\{[q(x,g(x)) \wedge \neg p(f(a)) \wedge [\forall z\, \neg q(x,z) \vee p(x)] \}\}$

Step 7: Move all universal quantifiers to the left

$\forall x \{\neg p(x) \vee \{[q(x,g(x)) \wedge \neg p(f(a))] \wedge [\forall z\, \neg q(x,z) \vee p(x)] \}\}$

$\forall x \forall z \{\neg p(x) \vee \{[q(x,g(x)) \wedge \neg p(f(a))] \wedge [\neg q(x,z) \vee p(x)] \}\}$

Step 8: Distribute $\wedge$・ over $\vee$.

$\forall x \forall z \{[\neg p(x) \vee q(x,g(x))] \wedge [\neg p(x) \vee \neg p(f(a))] \wedge [\neg p(x) \vee \neg q(x,z) \vee p(x)] \}$

Step 9: (Optional) Simplify

$\forall x \{[\neg p(x) \vee q(x,g(x))] \wedge \neg p(f(a)) \}$

Resolution

・If Unify(zj, ¬qk) = θ, then:

$z_1 \vee ... \vee z_m$, $q_1 \vee ... \vee q_n$

SUBST(θ,,$z_1 \vee ... \vee z_{i-1} \vee z_{i.1} \vee ... \vee z_m$

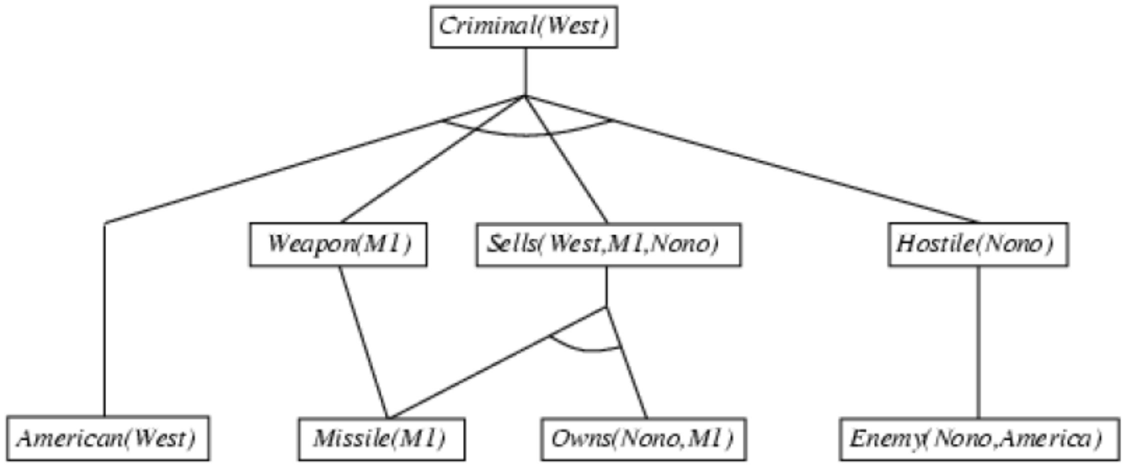
$\vee q_1 \vee ... \vee q_{k-1} \vee q_{k.1} \vee ... \vee q_n$)

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Convert the following sentences to predicate logic: 1. Marcus was a man. 2. Marcus was a Pompeian 3. All pompeians were Roman 4. Caesar was a ruler 5. All Romans were either loyal to Caesar or hated him. 6. Everyone is loyal to someone. 7. People only try to assassinate rulers they are not loyal to. 8. Marcus tried to assassinate Caesar.	June.2012	10

UNIT-02/LECTURE-07

forward chaining

Using a deduction to reach a conclusion from a set of antecedents is called forward chaining. In other words,the system starts from a set of facts,and a set of rules,and tries to find the way of using these rules and facts to deduce a conclusion or come up with a suitable couse of action. This is known as data driven reasoning.



The proof tree generated by forward chaining. Example knowledge base

•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.

•Prove that Col. West is a criminal

... it is a crime for an American to sell weapons to hostile nations:

American(x) $\wedge$ Weapon(y) $\wedge$ Sells(x,y,z) $\wedge$ Hostile(z) $\Rightarrow$ Criminal(x)

Nono … has some missiles, i.e., $\exists$x Owns(Nono,x) $\wedge$ Missile(x):

Owns(Nono,M1) and Missile(M1)

⋯ all of its missiles were sold to it by Colonel West

Missile(x) ∧ Owns(Nono,x) ⇒Sells(West,x,Nono)

Missiles are weapons:

Missile(x) ⇒Weapon(x)

An enemy of America counts as "hostile“:

Enemy(x,America) ⇒Hostile(x)

West, who is American …

American(West)

The country Nono, an enemy of America …

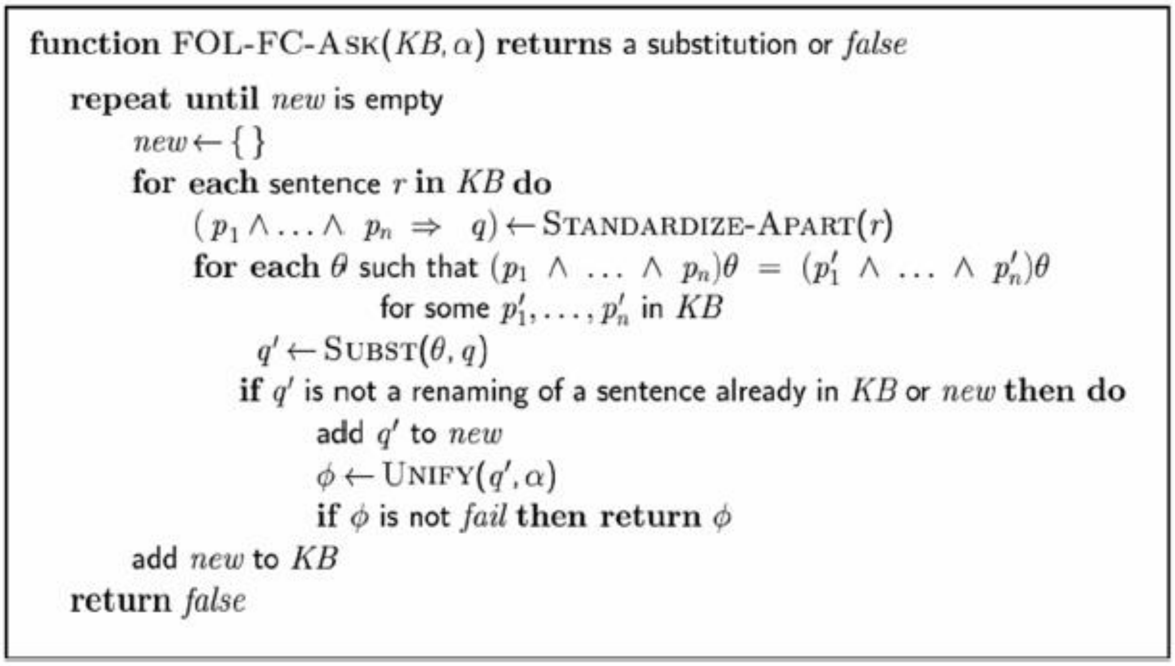
Enemy(Nono,America)

Note:

(a) The initial facts appear in the bottom level

(b) Facts inferred on the first iteration is in the middle level (c) The facts inferred on the 2nd iteration is at the top level

Forward chaining algorithm



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

backward chaining

Forward chaining applies a set of rules and facts to deduce whatever conclusions can be derived.

In backward chaining ,we start from a conclusion,which is the hypothesis we wish to prove,and we aim to show how that conclusion can be reached from the rules and facts in the data base. The conclusion we are aiming to prove is called a goal ,and the reasoning in this way is known as goal-driven.

Backward chaining example

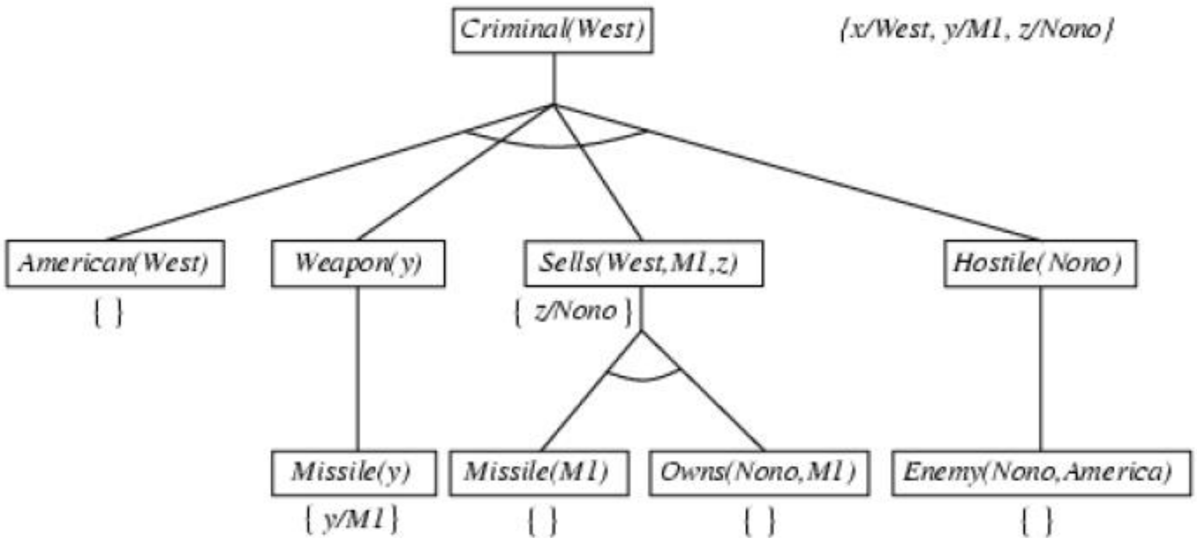


Fig : Proof tree constructed by backward chaining to prove that West is criminal.

Note:

(a) To prove Criminal(West) ,we have to prove four conjuncts below it.

(b) Some of which are in knowledge base, and others require further backward chaining.

Conjunctive normal form for first-order logic

As in the propositional case, first-order resolution requires that sentences be in **conjunctive normal form** (CNF)—that is, a conjunction of clauses, where each clause is a disjunction of literals.[6] Literals can contain variables, which are assumed to be universally quantified. For example, the sentence

$$\forall x \;\; American(x) \land Weapon(y) \land Sells(x,y,z) \land Hostile(z) \Rightarrow Criminal(x)$$

becomes, in CNF,

$$\neg American(x) \lor \neg Weapon(y) \lor \neg Sells(x,y,z) \lor \neg Hostile(z) \lor Criminal(x).$$

Every sentence of first-order logic can be converted into an inferentially equivalent CNF sentence. In particular, the CNF sentence will be unsatisfiable just when the original sentence is unsatisfiable, so we have a basis for doing proofs by contradiction on the CNF sentences. Here we have to eliminate existential quantifiers.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1		June.2011	10
Q.2		Dec.2012	10

Q.3		June.2011	10

UNIT-02/LECTURE-08

Deduction

An Inference Rule is a rule for obtaining a new formula [the consequence] from a set of given formulae [the premises].

A most famous inference rule is Modus Ponens:

{A, NOT A OR B}

B

For example:

{Sam is tall,

if Sam is tall then Sam is unhappy}

Sam is unhappy

When we introduce inference rules we want them to be Sound, that is, we want the consequence of the rule to be a logical consequence of the premises of the rule. Modus Ponens is sound. But the following rule, called Abduction , is not:

{B, NOT A OR B}

A

Theorem Proving:

The aim of theorem proving is to negate a theorem that is stated as a proposition. This could be done by finding inconsistency in the theorem. Theorem proving has been a basis for logic programming. Most propositions can be represented in Horn Clause format. When propositions are used for resolution, only restricted form (Headless horn form) can be used.

Horn Clause: Horn clause helps in representing procedures as specifications, which assist in automatic deduction of interested queries. Horn clause is either with headed or headless form. Headed form can have multiple propositions on the right hand side (called as antecedent) but can have only one atomic proposition on left side (called as consequent). Headless has no left side and it is used to state facts. Headless horn clauses are called as facts (or logical statement), while the headed horn clauses are called as rules.

Let's consider an example of sorting a list: Here, we specify the characteristics of a sorted list but not the actual process of rearranging a list.

sort(old_list, new_list) $\subset$permutation(old_list, new_list) $\cap$ sorted (new_list)

sorted(list) $\subset$ $\forall$x such that $1 \leq x < n$, list(x) $\leq$ list(x+1)

Then, we need to define the process of permutation:

permutation(old_list, new_list) $\subset \cdots$

Structures are represented using atomic proposition. A predicate with zero or more arguments is written in functional notation as functor(parameters). Fact is a term that is followed by a period and is similar to horn clause without a right-hand side. Here are some Headless Horn clauses (fact) examples, which have no conditions or right-hand part.

```
student(noraini).        % noraini is a student.
brother(arif, amri).     % arif is the brother of amri.
```

So, after specifying the facts above, if you generate a query as brother(arif,X), logical language responds X = amri. But if the query is given as brother(amri,X), logical language responds No.

Rule Statements: A rule is a term that is followed by :- (means if) and a series of terms separated by commas and ends with a period ".". The presence of the period indicates to the compiler or interpreter that the rule ends there. Headed Horn clause (Rule) has a head h, which is a predicate, and a body, which is a list of predicates p1, p2, … pn, written in any one of the following formats:

h $\leftarrow$ p1, p2, $\cdots$ pn

h $\Rightarrow$ p1, p2, $\cdots$ pn

h :- p1, p2, … pn

Means that h is true, only if p1, p2, … and pn are simultaneously true. Right side part is called as the antecedent (if part) and it may be single term or conjunction of terms. Left side is called as the consequent (then part) and it must be of one term only.

To make the logical language answer the brother(amri,X) query as X = arif, then we need to add a rule as below:

brother(X, Y) :- brother(Y,X), !.   % Please note the period at the end.

Legal goals or queries are made up of conjunctive propositions and/or propositions with variables. For example:

brother(amri, X).   % X is the variable to query for.

Theorem proving is a proposition represented in headless horn format and for this, we want the system to either prove or disapprove.

student(hafiz).   % Verify whether hafiz is a student or not.

Conjunction: Conjunction is the process of joining multiple terms with logical AND operations. Logical AND is represented using comma or $\wedge$. Implication is represented using :- or $\supset$. As an example, a person who could read as well as write is called as a literate. Here, read and write are the conditions that must be true, to imply that literate is true. By default, it is assumed that all the variables used are universal objects. This is represented as:

literate(C) :–read(C), write(C) % if read & write are true, then literate is true.

is equivalent to read(C) $\wedge$ write(C) $\supset$ literate(C)

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1		Dec.2010 Dec.2011 Dec.2012	5
Q.2		Dec.2009	10

UNIT-02/LECTURE-09

What is reasoning?

When we require any knowledge system to do something it has not been explicitly told how to do it must reason.

The system must figure out what it needs to know from what it already knows.

We have seen simple example of reasoning or drawing inferences already. For example if we know: Robins are birds.

All birds have wings. Then if we ask: Do robins have wings?

How can we reason?

To a certain extent this will depend on the knowledge representation chosen. Although a good knowledge representation scheme has to allow easy, natural and plausible reasoning. Listed below are very broad methods of how we may reason.

Formal reasoning

-- Basic rules of inference with logic knowledge representations.

Procedural reasoning

-- Uses procedures that specify how to perhaps solve (sub) problems.

Reasoning by analogy

-- Humans are good at this, more difficult for AI systems. E.g. If we are asked Can robins fly?. The system might reason that robins are like sparrows and it knows sparrows can fly so ...

Generalisation and abstraction

-- Again humans effective at this. This is basically getting towards learning and understanding methods.

Meta-level reasoning

-- Once again uses knowledge about about what you know and perhaps ordering it in some kind of importance.

Uncertain Reasoning?

Unfortunately the world is an uncertain place.
Any AI system that seeks to model and reasoning in such a world must be able to deal with this.

In particular it must be able to deal with:

- Incompleteness -- compensate for lack of knowledge.
- Inconsistencies -- resolve ambiguities and contradictions.
- Change -- it must be able to update its world knowledge base over time.

Clearly in order to deal with this some decision that a made are more likely to be true (or false) than others and we must introduce methods that can cope with this uncertainty.
There are three basic methods that can do this:

- Symbolic methods.
- Statistical methods.
- Fuzzy logic methods.

Non-Monotonic Reasoning

Predicate logic and the inferences we perform on it is an example of monotonic reasoning.
In monotonic reasoning if we enlarge at set of axioms we cannot retract any existing assertions or axioms.
Humans do not adhere to this monotonic structure when reasoning:

- we need to jump to conclusions in order to plan and, more basically, survive.
  - ✓ we cannot anticipate all possible outcomes of our plan.
  - ✓ we must make assumptions about things we do not specifically know about.

Traditional systems based on predicate logic are monotonic. Here number of statements known to be true increases with time. New statements are added and new theorems are proved, but the previously known statements never become invalid.

In monotonic systems there is no need to check for in consistencies between new statements and the old knowledge. When a proof is made, the basis of the proof need not be remembered, since the old statements never disappear. But monotonic systems are not good in real problem domains where the information is incomplete, situations change and new assumptions are generated while solving new problems.

Non monotonic reasoning is based on default reasoning or “most probabilistic choice”. S is assumed to be true as long as there is no evidence to the contrary. For example when we visit a friend’s home, we buy biscuits for the children . because we believe that most children like biscuits. In this case we do not have information to the contrary. A computational description of default reasoning must relate the lack of information on X to conclude on Y.

Default reasoning (or most probabilistic choice) is defined as follows:

Definition 1 : If X is not known, then conclude Y.

Definition 2 : If X can not be proved, then conclude Y.

Definition 3: If X can not be proved in some allocated amount of time then conclude Y.

It is to be noted that the above reasoning process lies outside the realm of logic. It conclude on Y if X can not be proved, but never bothers to find whether X can be proved or not. Hence the default reasoning systems can not be characterized formally. Even if one succeeds in gaining complete information at the moment, the validity of the information may not be for ever, since it is a

changing world. What appears to be true now, may be so at a later time (in a non monotonic system).

One way to solve the problem of a changing world to delete statements when they are no longer accurate, and replace them by more accurate statements. This leads to a non monotonic system in which statements can be deleted as well as added to the knowledge base. When a statement is deleted, other related statements may also have to be deleted. Non monotonic reasoning systems may be necessary due to any of the following reasons.

- The presence of incomplete inforamtio0n requires default reasoning.

- A changing world must be described by a changing database.

- Generating a complete solution to a problem may require temporary
  assumptions about partial solutions.

Non monotonic systems are harder to deal with than monotonic systems. This is because when a statement is deleted as “no more valid”, other related statements have to be backtracked and they should be either deleted or new proofs have to be found for them. This is called dependency directed backtracking (DDB). In order to propagate the current changes into the database, the statements on which a particular proof depends, should also be stored along with the proof. Thus non – monotonic systems require more storage space as well as more processing time than monotonic systems.

Default reasoning

This is a very common from of non-monotonic reasoning. Here We want to draw conclusions based on what is most likely to be true.
We have already seen examples of this and possible ways to represent this knowledge.
We will discuss two approaches to do this:

- Non-Monotonic logic.
- Default logic.

DO NOT get confused about the label Non-Monotonic and Default being applied to reasoning and a particular logic. Non-Monotonic reasoning is generic descriptions of a class of reasoning. Non-Monotonic logic is a specific theory. The same goes for Default reasoning and Default logic.

Non-Monotonic Logic

This is basically an extension of first-order predicate logic to include a modal operator, M. The purpose of this is to allow for consistency.

For example: $\forall x$: plays_instrument(x) $\wedge M$ improvises(x) $\rightarrow$ jazz_musician(x)

states that for all x is x plays an instrument and if the fact that x can improvise is consistent with all other knowledge then we can conclude that x is a jazz musician.

How do we define consistency?

One common solution (consistent with PROLOG notation) is
to show that fact P is true attempt to prove $\neg P$. If we fail we may say that P is consistent (since $\neg P$ is false).

However consider the famous set of assertions relating to President Nixon.

$\forall x$: Republican(x) $\wedge M \neg$ Pacifist(x $\rightarrow \neg$ Pacifist(x)

$\forall x$: Quaker(x) $\wedge M$ Pacifist(x) $\rightarrow$ Pacifist(x)

Now this states that Quakers tend to be pacifists and Republicans tend not to be.

BUT Nixon was both a Quaker and a Republican so we could assert:

Quaker(Nixon)

Republican(Nixon)

This now leads to our total knowledge becoming inconsistent.

Default Logic

Default logic introduces a new inference rule:

$$\frac{A:B}{C}$$

Which states if A is deducible and it is consistent to assume B then conclude C.

Now this is similar to Non-monotonic logic but there are some distinctions: New inference rules are used for computing the set of plausible extensions. So in the Nixon example above Default logic can support both assertions since is does not say anything about how choose between them -- it will depend on the inference being made.In Default logic any nonmonotonic expressions are rules of inference rather than expressions.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Explain monotonic and non-monotonic reasoning with example	June.2013 June.2006	10
Q.2	Differentiate the monotonic and non-monotonic reasoning	June.2011	10

REFERENCE

BOOK	**AUTHOR**	**PRIORITY**
Artificial Intelligence.	Elaine Rich & Kevin Knight	1
Artificial Intelligence. A modern approach	Russell Stuart & peter Norvig	2