6 First Order Logic - Predicate Logic

Syllabus

First Order Logic : *Representation Revisited, Syntax and Semantics of First Order logic, Using First Order logic.*

Inference in First Order Logic : *Propositional Versus First Order Inference, Unification, Forward Chaining, Backward Chaining, Resolution.*

Contents

6.1 First Order Logic - [Predicate Logic]

GTU : Winter-18, Summer-20

6.1.1 Introduction to First Order Logic

It is a expressive language which has well defined syntax and semantics. It uses three things to represent the world model :

1) Objects :

People, houses, numbers, theories, colours, games, wars etc. These are nouns and nounphrases in natural languages.

2) Relations :

Red, round, prime, brother, bigger than, inside, owns, comes between etc. These are associations among objects or properties of objects. Relations can be unary such as green or n-ary such as brother_of.

3) Function :

Father_of, best_friend, one_more_than etc.

It is a relation in which there is only one "value" for a given input.

For example : "Two plus five equals seven".

Objects - Two, five, seven.

Relation - equals.

Function - plus.

("Two plus five" is the name for the object that is obtained by applying the function plus to the objects two and five).

For example : "Multitalented King Krishna Ruled Dwarika."

Objects - Krishna, Dwarika.

Relation - Ruled.

Functions - King, Multitalented.

6.1.2 Properties of First Order Logic

1. It has ability to represent facts about some or all of the objects in the universe.
2. It enables to represent law and rules extracted from real world.
3. It is useful language representation in mathematics, philosophy and AI related fields.
4. It represents facts in more realistic manner rather than just the true or false statement.

5. First order logic makes ontological commitment.

   Ontological commitment means what assumptions language makes about the nature of reality. First order logic assumes that real world consists of objects and certain relationship they hold or not among them.

6.1.3 First Order Logic-Extension to Propositional Logic

First order logic extends propositional logic in two directions :

1) It provides an inner structure for sentences. They are viewed as expressing relations between objects or individuals.
2) It provides a means to express, and reason with, generalizations. It makes it possible to say that a certain property holds for objects, or for some objects, or for no object.

6.1.4 Variations of First Order Logic

1. Temporal logic :

   It assumes that facts hold at some particular times. These times also have orders.

2. High order logic :

   It views the relations and functions referred to by first order logic as objects in themselves.

6.1.5 Characteristics of Logic

1) Epistemological commitments :

   It is ability to show, the possible states of knowledge that it allows with respect to each fact.

   From AI agent point of view, any sentence can be true, false or has no opinion states depending on the knowledge base.

2) Probability theory :

   It is ability to assign a degree of probability to the sentence ranging from 0 to 1. It can be a belief probability for agent.

   A high probability sentence would be a better choice for AI agent when it needs to think.

6.1.6 Formal Languages and their Ontological and Epistemological Commitments

Language	Ontological commitment (what exists in the world)	Epistemological commitment (What an agent belives 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

6.1.7 Higher-Order Logic

- FOPL is called first-order because it allows quantifiers to range over objects (terms) but not properties, relations or functions applied to those objects.
- Second-order logic allows quantifiers to range over predicates and functions as well :

  "X" Y [(x=y) U ("p p(x) U" p(y))]

  Says that two objects are equal if and only if they have exactly the same properties.

  "f" g [(f=g) U ("X f(x) = g(x))]

  Says that two functions are equal if and only if they have the same value for all possible arguments.
- Third-order would allow quantifying over predicates of predicates, etc.
  For example : A second-order predicate would be symetric (p) stating that a binary predicate p represents a symmetric relation.
- Higher order logic views the relations and functions referred by first-order logic as objects in themselves.
- Higher order logic makes further ontological commitments.

6.1.8 Syntax for First Order Logic

1. Model :

Model of logical language are the formal structures that constitute the possible world under consideration.

In first order language, models have objects in them.

2. Domain of a model :

It is a non-empty set of objects in model. These objects are also referred to as domain elements.

3. Relation :

It is association among objects. It is represented as a tuple. It is a set of tuples of objects that are related.

Note *Tuple is collection of object in some specific order, written inside angle brackets.*

4. Function :

Is a special type of relation in that a given object must be related to exactly one object in this way.

5. Symbols :

Smallest syntatic element in first order logic are symbols, that can stand for objects, relations, functions.

- **Following are the first order logic's symbols :**

1. **Truth symbols :**

   Truth symbols true and false. These are reserved symbols.

2. **Constant symbols :**

   Constant symbols are symbol expressions having the first character lowercase, they stand for objects.

   Example : Cindrella, A, B, Ram, Red, etc.

3. **Variable symbols :**

   These are symbol expressions beginning with an uppercase character and designate unspecified objects.

   Example : X, Y, Z, etc.

4. **Predicate symbols :**

   Predicate symbols are symbols beginning with a lowercase letter and they stand for relations.

   Example : Stepsister, likes, color, etc.

5. **Function symbols :**

   Function symbols are symbol expressions having the first character lowercase and they stand for functions. Functions have an attached arity indicating the number of elements of the domain mapped onto each element of the range.

A function expression consists of a function constant of arity n, followed by 'n' terms $t_1, t_2, t_3 .... t_n$ enclosed in parentheses and separated by commas.

Example : RightLegShoe, FatherOF, ColorOF etc.

- **A example depicting various synatic terms used in first order logic :**

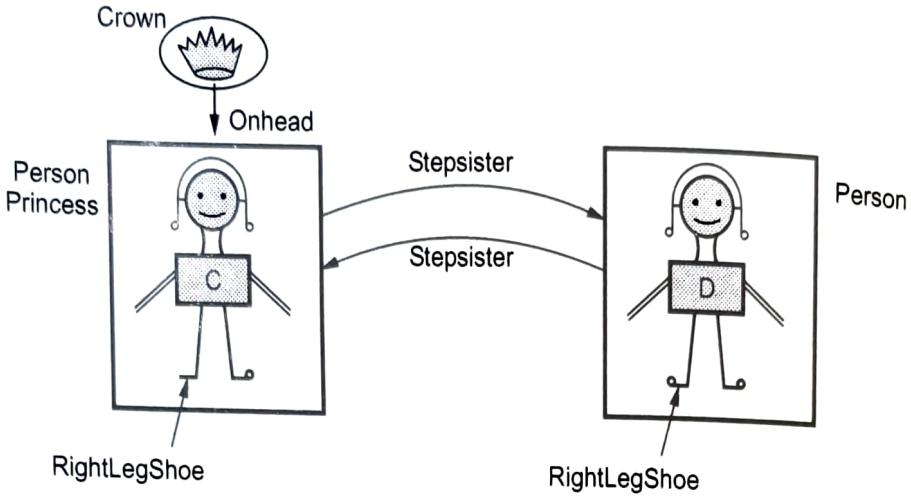


Fig. 6.1.1 Cindrella and Drizella are sisters

Objects :

1. Person princess Cindrella.
2. Person Drizella.
3. Crown.
4. Right Leg shoe of Cindrella.
5. Right Leg shoe of Drizella.

Relation :

1. "Onhead" < the Crown, Princess Cindrella >
2. "Stepsister" < Cindrella, Drizella >
3. "Person" < Cindrella >
4. Person < Drizella >
5. Princess < Cindrella >

Function :

[Every person wears shoe in right leg]

1. < Cindrella the Princess > → Rightlegshoe.
2. < Drizella > → Rightlegshoe

6.1.9 Sentence in First Order Logic

1) Terms

i. It is a logical expression that refer to an object or individuals.

ii. Terms are built using constant, variable, and function symbols.

iii. Constant symbols are terms.

iv. Functions are terms.

For example : LeftLegShoe (Cindrella), cat, X, times (2,3), blue, Mother (Ram), Seeta.

2) Atomic sentences

i. An atomic sentence is formed from a predicate symbol followed by a parenthesized list of terms.

For example : Stepsister (Cindrella, Drizella)

ii. Atomic sentences can have complex terms as the arguments.

For example : Married (Father (Cindrella), Mother(Drizella))

iii. Atomic sentences are also called atomic expressions, atoms or propositions.

For example : Equal (plus (two, three), five) is an atomic sentence.

3) Predicates

i. Predicates have a value of true or false.

ii. A predicate can take arguments, which are terms.

iii. A predicate with one argument expresses a property of an object.

For example : Student (Dipu)

iv. A predicate with two or more arguments expresses a relation between objects.

For example : Likes (Dipu, Books), Likes (Dipu, School-of (Dipu)).

v. A predicate with no arguments is a simple proposition, as in propositional logic.

4) Complex sentences

i. Atomic sentences can be connected to each other to form complex sentence.

Logical connectives, $\wedge$, $\vee$, $\neg$, $\rightarrow$ can be used to connect atomic sentences.

For example :

$\neg$ Princess (Drizella) $\rightarrow$ Princess (Cindrella)

ii. (foo (two, two, plus (two, three))) $\rightarrow$ (equal (plus (three, two), five) $\equiv$ true) is a sentence because all its components are sentences, appropriatly connected by logical operators.

- **Various sentences in first order logic formed using connectives :**
  1) If S is a sentence, then so is its negation, $\neg S$.
  2) If S_1, and S_2 are sentences, then so is their conjunction, $S_1 \wedge S_2$.
  3) If S_1 and S_2 are sentences, then so is their disjunction, $S_1 \vee S_2$.
  4) If S_1 and S_2 are sentences, then so is their implication, $S_1 \rightarrow S_2$.
  5) If S_1 and S_2 are sentences, then so is their equivalence, $S_1 \equiv S_2$.

6.1.10 Semantic of First Order Logic

Once constant symbols, relations and functions are decided, one need the interpretations to relate various symbols in first order logic. There are many ways to interpret the relationships as describe below : -

Interpretation

Let the domain D be a nonempty set.

An interpretation over D, is an assignment of the entities of D to each of the, constant, variable, predicate and function symbols of a predicate calculus expression such that :

1) Each constant is assigned an element of D.
2) Each variable is assigned to a non-empty subset of D; these are the allowable substitutions for that variable.
3) Each function 'f' of arity 'm' is defined on m arguments of D and defines a mapping from D^m into D.
4) Each predicate 'p' of arity 'n' is defined on 'n' arguments from D and defines a mapping from D^n into {T, F}.

- Given an interpretation, the meaning of an expression is a truth value assignment over the interpretation.

Truth value of first order logic expression

Assume an expression E and an interpretation I for E over a non-empty domain D. The truth value for E is determined by : -

1) The value of a constant is the element of D it is assigned to by I.
2) The value of a variable is the set of elements of D it is assigned to by I.
3) The value of a function expression is that element of D obtained by evaluating the function for the parameter values assigned by the interpretation.
4) The value of truth symbol "true" is T and "false" is F.
5) The value of an atomic sentence is either T or F, as determined by the interpretation I.

6) The value of the negation of a sentence is T if the value of the sentence is F and it is F if the value of the sentence is T.

7) The value of the conjunction of two sentences is T if the value of both sentences is T and it is F otherwise.

8) The truth value of expressions using $\vee$, $\rightarrow$, and $\equiv$ is determined from the value of their operands same as in propositional logic.

6.1.11 Quantifiers

They are used for expressing properties of entire collection of objects, rather than just a single object. We need complex sentences to represent huge amount of data. This will really help to represent real world data which has complex association among the objects.

There are two quantifiers in first order logic : -

1) **Universal Quantifier**
2) **Existential Quantifier**

6.1.11.1 Universal Quantifier (∀)

i) It is represented using symbol (∀)

∀ is pronounced as "for all".

ii) The sentence formed using universal quantifier use a variable.

- **Variable :**
  1. It is term itself.
  2. It is represented using lowercase alphabets like x, y, z.
  3. It can take up value from allowable objects, relation or function set.
  4. A term without variable is called as ground term.

iii) The sentence

∀ x P

Say that P is true for every object x, where P is a logical expression.

For example :

a) "All Princess are Person" can be represented as,

$\forall$ x Princess (x) $\Rightarrow$ Person (x)

b) "If the universe of discourse is people, then this means that everyone is happy" can be represented as,

$\forall$ x Happy (x)

iv) '∀' can make statement about every object.

v) **'⇒' is natural connective to use with '∀'.**

vi) **Common mistake to avoid :** Do not use ∧ as the main connective with ∀.

For example :

∀ x At (x, IIT) ∧ Smart (x)

It means that "Everyone is at IIT and everyone is smart."

It is mistake because you wanted to say "everyone at IIT is smart."

6.1.11.2 Existential Quantifier (∃)

i) These are used to make statement about some objects and not just for all.

ii) The objects are not named.

iii) The sentence, ∃ x P says that P is true for atleast one object x.

For example :

a) "Princess Cindrella has crown on her head" can be expressed as,

∃x Crown (x) ∧ OnHead (x, Cindrella)

[Crown is a crown and it is on Cindrella's head]

b) "If the universe of discourse is people, then this means there is at least one happy person", can be expressed as,

∃ x Happy (x)

iv) '∃ x' it is pronounced as **"There exists an x such that...."**

OR

"For some x....".

v) **'∧' is natural connective to be used with '∃'.**

vi) **Common mistake to avoid :** Do not use ⇒ as the main connective with ∃.

For example :

∃ x At (x, IIT) ⇒ Smart (x) is true if there is anyone who is not at IIT.

- **First order logic sentences using quantifier :**
  1) If X is a variable and S is a sentence, then ∀ X S is a sentence.
  2) If X is a variable and S is a sentence, then ∃ X S is a sentence.

6.1.11.3 Truth Values of First Order Logic Expressions Containing Quantifiers

For a variable X and a sentence S containing X :

1) The value of ∀ X S is True if S is True for all assignments to X under I, and it is False otherwise.

2) The value of ∃ X S is True if there is an assignment to X in the interpretation under which S is True; otherwise it is False.

6.1.11.4 The Nested Quantifiers

Using multiple (nested quantifier) we can express complex sentences. We can do nesting of same or different quantifiers.

- [Simple nesting using same quantifier ∀]
  For example : "Sisters are siblings" it can be written as,
  $\forall x\ \forall y$ Sisters $(x,y) \Rightarrow$ Siblings (x,y).
- [Consecutive quantifier of the same type can be written as one quantifier with several variables]
  For example : "Siblinghood" is symmetric relationship then we have,
  $\forall x,y$ Sibling $(x,y) \Leftrightarrow$ Sibling (y,x).
- [Mixing of two different quantifiers]
  For example : -
  a) Everybody loves somebody.
  $\forall x \exists y$ Loves (x,y)
  b) There is someone who is loved by everybody.
  $\exists x \forall y$ Loves (y,x)

Note *The order of quantifier is important. Change in the order effectively changes the meaning of predicate.*

In above example :
$\forall y \exists x$ Loves (y,x)
Means that, for every y there is somebody to Love (!!!) (?)

6.1.11.5 Relationship between ∀ and ∃

Two quantifiers are connected with each other through "negation",
Saying : "All girls like Rose" means that
"There is no girl who does not like Rose".
$\forall x$ Like (x, Rose) is equivalent to
$\neg \exists x \neg$ Like (x, Rose).

- **DeMorgan's rule for quantifiers**

1) $\forall x \neg P \equiv \neg \exists x\ P$
2) $\neg P \wedge \neg Q \equiv \neg (P \vee Q)$
3) $\neg \forall x\ P \equiv \exists x \neg P$
4) $\neg (P \wedge Q) \equiv \neg P \vee \neg Q$
5) $\forall x\ P \equiv \neg \exists x \neg P$
6) $P \wedge Q \equiv \neg (\neg P \vee \neg Q)$

7) $\exists x\ P \equiv \neg\ \forall x \neg\ P$

8) $P \vee Q \equiv \neg\ (\neg P \wedge \neg\ Q)$

6.1.11.6 Equality

1) In first order logic equality symbol is used to make sentences that can associate two more atomic sentences.
2) It can be used to state fact about given function.
3) It can be used with negation to express that two terms are not the same object.

For example : Stepsister (Cindrella) = X

To say "Cindrella has at least 2 step-sisters"

we can write $\exists$ x,y Stepsister (x, Cinderella) $\wedge$ Stepsister (y, Cindrella) $\wedge \neg$ (x=y)

6.1.11.7 Properties of Quantifiers

1) Quantifiers of same type commute

i) $\forall$ x $\forall$ y is same as $\forall$ y $\forall$ x

ii) $\exists$ x $\exists$ y is same as $\exists$ y $\exists$ x.

For example :

a) Statement 1) $\forall x \forall y$ father (x,y) $\rightarrow$ Parent (x,y)

Statement 2) $\forall y \forall x$ father (x,y) $\rightarrow$ Parent (x,y)

Note that, Both statement 1 and statement 2 are same.

b) Statement 1) $\exists x \exists y$ Likes (x,y) $\rightarrow$ FriendOF (x,y)

Statement 2) $\exists y \exists x$ Likes (x,y) $\rightarrow$ FriendOF (x,y)

Note that, Both statement 1 and statement 2 are same.

2) Quantifiers of different type do not commute

$\exists x \forall y$ **is not the same as** $\forall y \exists x$

For example :

a) Statement 1 :

$\exists x \forall y$ Loves (x,y)

"There is a person who loves everyone in the world".

Statement 2 :

$\forall y \exists x$ Loves (x,y)

"Everyone in the world is loved by at least one person".

Note that,

Statement 1 and Statement 2 are not same.

b) Statement 1 :

$\forall x \exists y$ Mother (x,y)

"Everyone has a mother" (correct)

Statement 2 :

$\exists y \forall x$ Mother (x,y)

"There is a person who is the mother of everyone". (Wrong)

3) Quantifier Duality

$\forall x$ Likes (x, IceCream) is the same as

$\neg \exists x \neg$ Likes (x, IceCream).

$\exists x$ Likes (x, Brocoli) is the same as

$\neg \forall x \neg$ Likes (x, Brocoli)

6.1.11.8 The Examples Bank

Here are few varieties of first order logic sentences : -

1) $\underbrace{\underbrace{\text{StepSister}}_{\text{Predicate}} \; \underbrace{(\underbrace{\text{Drizella}}_{\text{Constant}}}_{\text{Term}}, \; \underbrace{\underbrace{\text{Cindrella}}_{\text{Constant}})}_{\text{Term}}}_{\text{Atomic sentence}}$

2) $\underbrace{\underbrace{>}_{\text{Predicate}} \; [(\underbrace{\underbrace{\text{Length}}_{\text{Function}} \; (\underbrace{\underbrace{\text{Leftshoeof}}_{\text{FunctionOF}} \; \underbrace{(\underbrace{\text{Cindrella}}_{\text{Constant}})}}_{\text{Term}}}_{\text{Term}}, \; \text{Length(Leftshoeof (Drizella)))]}}_{\text{Atomic sentence}}$

3) $\underbrace{\underbrace{\underbrace{\text{Sibling}}_{\text{Predicate}} \; (\underbrace{\underbrace{\text{Cindrella}}_{\text{Term}}, \; \underbrace{\text{Drizella})}_{\text{Term}}}_{\text{Atomic sentence}} \Rightarrow \underbrace{\text{Sibling(Drizella, Cindrella)}}_{\text{Atomic sentence}}}_{\text{Complex sentence}}$

4) Everyone studying in India is smart.

$\underbrace{\forall x}_{\text{Variable}} \; \underbrace{(\text{StudiesIn } (x, \text{India}) \Rightarrow \text{Smart } (x))}_{\text{Sentences}}$

5) Someone studying in India is smart.

$\exists x$ (StudiesIn (x, India) $\wedge$ Smart (x))

6) "Ram has an umbrella"can be written as,

$\exists y$ (Has (Ram, y) $\wedge$ IsUmbrella (y))

7) "Anything that has an umbrella is not wet".

$\forall x [(\exists y (Has (x,y) \wedge IsUmbrella (y))) \Rightarrow \neg (IsWet (x))]$

8) "Any person who has an umbrella is not wet".

$\forall x [IsPerson (x) \Rightarrow ((\exists y (Has (x,y) \wedge IsUmbrella (y))) \Rightarrow \neg (IsWet (x)))]$

9) Ram has at least two umbrellas.

$\exists x [\exists y (Has (Ram, x) \wedge IsUmbrella (x) \wedge Has (Ram, y) \wedge IsUmbrella (y) \wedge \neg (x=y)]$

10) Everyone Has umbrella.

$\forall x \exists y [Has (x,y) \wedge IsUmbrella (y)]$

11) If it doesn't rain on Monday, Nimu will go to the school.

$\neg$ Weather (rain, Monday) $\rightarrow$ Go (Nimu, School)

12) Sweety is a Doberman pinscher and a good dog.

GoodDog (Sweety) $\wedge$ Isa (Sweety, Doberman)

13) All basketball players are tall.

$\forall x$ (Basketball_Player (x) $\rightarrow$ Tall (x))

14) Some people like cricket.

$\exists x$ (Person (x) $\wedge$ Likes (x, cricket))

15) If wishes were horses, beggars would ride.

Equal (wishes, horses) $\rightarrow$ Ride (Beggars)

16) Nobody likes taxes.

$\neg \exists$ x Likes (x, taxes)

17) Sisters are siblings.

$\forall$ x, y [Sisters (x,y) $\Rightarrow$ Siblings (x,y)]

Note *Sibling is symmetric relationship.*

18) Siblings is symmetric.

$\forall$ x, y (Sibling (x, y) $\Leftrightarrow$ Sibling (y, x)]

19) One's mother is one's female parent.

$\forall$ x, y (Mother (x, y) $\Leftrightarrow$ (Female (x) $\wedge$ Parent (x, y)))

20) A first cousine is a child of a parent's sibling.

$\forall$ x, y [FirstCousin (x, y) $\leftrightarrow \exists$ p,

ps (Parent (p,x) $\wedge$ Sibling (ps, p) $\wedge$ Parent (ps, y))]

First order logic semantic example

"A Blocks World".

1. We can model a blocks world, for a control algorithm written for a robotic arm.
2. The diagramatic representation is as shown below :

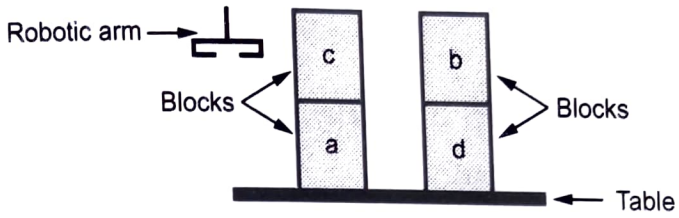


Fig. 6.1.2 The blocks world

First order logic expressions for above situation of block world :

1) On (c, a)
2) On (b, d)
3) On Table (a)
4) On Table (d)
5) Clear (c) [Clear says that there is no block above argument block]
6) Clear (b)
7) hand_empty.

6.1.12 Using First Order Logic for Knowledge Base

1) Sentences are added to knowledge base using **TELL** interface.

   Such sentences are called **"assertions"**.

   For example : TELL (KB, Princess (Cindrella))

2) i) When we want to query knowledgebase then ASK interface is used.
   ii) Questions asked to knowledgebase are called as queries or goals.

   For example : ASK (KB, Person (Cindrella)) will return true.

   iii) As an answer to a query knowledge base can return true or false.
   iv) We can use quantifiers in query. But one need to **bind** or **substitute** variables in sentence so as to get fruitful output.

   For example : ASK (KB, $\exists$x Person (x))

   It will return true but **"Who is Person" is more informative** than just that **"Somebody is person"**.

6.1.13 The Kinship Domain

This is a type of relationship which represent real life family relationships like FatherOF, MotherOF, SonOF, etc.

For example :

Rajiv is SonOF Indira.

Rajiv's grandfather is FatherOF Indira.

In Kinship domain relationships -

i) Objects - are people.

ii) We can have unary predicates like Male, Female.

iii) Relations are parenthood, Marriage.

These are represented using binary predicates like Parent, Siblings, Aunt, Uncle, Grandfathers, etc.

For example :

1) One's wife is one's female spouse $\forall$ w, h wife (w, h) $\Leftrightarrow$ (female (w) $\wedge$ Spouse (w,h))

2) Male and female are disjoint categories $\forall$ x Male (x) $\Leftrightarrow \neg$ Female (x)

3) Parent and child is inverse relation $\forall$ p, c Parent (p, c) $\Leftrightarrow$ Child (c, p)

iv) Sentences in Kinship domain are called as axioms (rules) or definitions.

v) Some axioms can be theorem i.e. they are derived from other axioms

$\forall$ x, y Sibling (x, y) $\Leftrightarrow$ Sibling (y,x).

6.1.14 Number, Sets and Lists

Number, sets, list are the means with which a knowledge base can be easy to build, because numbers, sets, list have symbols, predicates and functions that can be utilized for representing knowledge; A brief idea is given in following section :

- **Numbers** :

Using numbers one can represent large theory statements, which can be useful for AI agent.

For example : To describe natural number theory

1) We need predicate NatNum. It will be true for all natural numbers.

2) We need constant symbol 0 (zero)

3) We need successor function symbol S.

4) There is axiom called as Peano Axiom that define natural number and addition.

5) We can define any natural number using following predicates,

NatNum (0)

$\forall$ n NatNum (n) $\Rightarrow$ NatNum (S(n))

i.e. we can derive from O any natural number using successor function i.e three can be derived from O as,

$$S(\underbrace{S(\underbrace{S(O)}_{1})}_{2})}_{3}$$

6) We can have constraints on function like

i) $\forall$ n O $\neq$ S(n)

i.e. O can not be a successor of any 'n' as 'O' is first natural number.

ii) $\forall$ m, n $\quad m \neq n \Rightarrow S(m) \neq S(n)$

7) We can define addition as

$\forall$ m NatNum (m) $\Rightarrow$ + (m, 0) = m

[It means that adding 0 to any natural number gives that number].

$\forall$ m, n NatNum (m) $\wedge$ NatNum (n) $\Rightarrow$ + (S(m), n) = S (+(m, n))

8) Ordinary mathematics have **infix** notations for expression like m+n (operator is between operands)

9) In first order logic we use notation +mn which is called **prefix** (where operator comes first).

- **Sets :**

Set is collection of unordered distinct elements. We can use number theory to represent set theory. Set theory is also useful for AI agent, which can be used for storing data.

For example :

i) We can use unary predicate set which is true for set.

ii) Binary predicate

a) $x \in S$ (x is member of set S)

b) $S_1 \subseteq S_2$ (S_1 is subset of S_2)

Lists :

List is a collection of ordered elements and a element can appear repetedly in the list.

[Lisp is a List programming language which makes use of lists and predicates and function on those list]

For example :

- Nil is constant list with no elements.
- Append - Add element at last in list.
- First - Return first element of list are functions.
- Find is a predicate which search element in list.

6.1.15 Synchronic and Diachronic Sentences

1) The sentences dealing with time are synchronic sentences. They relate properties of a world state to other properties of the same world state.
2) The sentences that allow reasoning "across time" are called as diachronic sentences. That is previous state combined with current action should provide reasoning to determine its current location.

6.1.16 Synchronic Rules for Inferencing

6.1.16.1 Diagnostic Rules

1) From observed effect they generate hidden causes.
2) They help to deduce hidden facts in the world.

For example : Consider Wumpus world.

Representing Wumpus world environment

1) Various objects in Wumpus world would be squares, pits, the Wumpus, etc.
2) Each square should be named. The fact about adjacent squares should be specified.
3) Pit can be represented using unary predicate, that is true for squares containing pit.
4) A single Wumpus in wumpus world is represented using constant Wumpus. As Wumpus lives in one square its location can be described using functions like Home (Wumpus) etc.
5) By having knowledge about places and their properties agent can infer where is pit and where is Wumpus.
6) The agent will keep on changing its location over time which can be described using function like,

   At (Agent, s, t) means that agent is square 's' at time 't'.

Diagnostic rule for finding Pit is,

A) "If square is breezy some adjacent square must contain pit", which is written as,

$\forall$ s Breezy (s) $\Rightarrow \exists$ r Adjacent (r, s) $\wedge$ Pit (r).

B) "If square is not breezy, no adjacent square contains pit", which is written as,

$\forall$ s $\neg$ Breezy (s) $\Rightarrow \neg \exists$ r Adjacent (r, s) $\wedge$ Pit (r)

C) Combining A and B clause we get biconditional sentence,

$\forall$ s Breezy (s) $\Leftrightarrow \exists$ r Adjacent (r, s) $\wedge$ Pit (r)

6.1.16.2 Causal Rules

1) These rules reflect the assumed direction of causality in the world.
2) Some hidden property (fact) of the world causes certain percepts to be generated. For example : A pit causes all adjacent squares to be breezy.
   A. Which is written as

      $\forall$ r Pit (r) $\Rightarrow$ [$\forall$ s Adjacent (r, s) $\Rightarrow$ Breezy (s)].

   B. If all squares adjacent to a given square are pitless, the square will not be breezy, which is written as,

      $\forall$ s [$\forall$ r Adjacent (r, s) $\Rightarrow \neg$ Pit (r)] $\Rightarrow \neg$ Breezy (s).

   C. A and B can be combined together to form biconditional sentence. Biconditional sentences are causal in nature because from world state they can generate facts.

6.1.16.3 Model Based Reasoning Systems and Diagnostic Reasoning Systems

1) Systems that reason with causal rules are called model-based reasoning systems.

   For example : In Medical Systems where disease needs to be investigated one can start from diseases process to reach at the conclusion about infected disease. Symptoms are used as facts about current state rather than the direct reason for inference.

2) Systems that directly makes association between facts and conclusion are diagnostic reasoning systems.

   For example : In finding infection of the diseases a direct association is made between the symptoms and conclusion about disease.

3) The rule of thumb is -

   Agent designer should concentrate on getting knowledge right without worrying about inferencing procedure.

   More correct knowledge leads to better inferencing.

Any complete logical inferencing algorithm will infer the strongest possible description of the world state provided that,

i) It receives all possible available percepts.

ii) If the axioms are correctly and completely described exactly the way world works and the way the percepts are produced in the world.

6.2 The Wumpus World Represented using First Order Logic

6.2.1 Wumpus World Revisited

1) Agent receives percept with five elements shown as follows : -

   Percept ([Stench, Breeze, Glitter, None, None] , 5)

   Here percept is binary predicate and stench etc. are constants stored in the list.

2) The knowledge base must contain both the percept and the time at which it occured. We will use integer for time steps.

3) The actions in Wumpus world can be represented using logical terms.

   Turn (Right), Turn (Left), Forward, Shoot, Grab, Release, Climb.

   To choose best action from action list the query can be constructed as,

   $\exists$ a BestAction (a, 5).

6.2.2 Reasoning in Wumpus World using First Order Logic

1) We need percept data for reasoning.

2) Raw percept data implies certain facts about the current state.

   For example :

   $\forall$ t, s, g, m, c Percept ([s, Breeze, g, m, c], t) $\Rightarrow$ Breeze (t)

   $\forall$ t, s, b, m, c Percept ([s, b, Glitter, m, c], t) $\Rightarrow$ Glitter (t).

3) These rules exhibit **a trivial form of the reasoning process** called **perception**.

4) Simple reflex behavior can also be implemented using quantified implication sentences.

   For example : $\forall$ t Glitter (t) $\Rightarrow$ BestAction (Grab, t)

5) Given all above axioms the 'BestAction' query would return the best action as 'Grab'.

6.3 Knowledge Engineering in First Order Logic

The process of constructing knowledge base is called as knowledge base engineering. These knowledge base engineering projects vary widely in content, scope and difficulty. A specially designated person called as knowledge engineer will do task of knowledge engineering. He is a person who,

1) Investigates domain.
2) Understands which concepts are important in that domain.
3) Creates format for representing objects and relations.

6.3.1 We have Two Types of Knowledge Base

1) General purpose knowledge base : Which is intended to support queries about full range of human knowledge. In this, we can expect any kind of query which knowledge base will have to infer.
2) Special purpose knowledge base : Which has restricted domain [which can be a certain problem specific domain].

   Here expected queries are known in advance.

6.3.2 Steps in Knowledgebase Engineering Process

1) Identify the task :

i. The knowledgebase engineer must find the range of questions that knowledgebase will support. He should also find the facts that would be available for each specific problem instance.

ii. Once the task is identified it will determine what knowledge must be represented in order to relate problem instances to answers.

iii. Note that identifying the task is similar to PEAS designing.

iv. For example : In Wumpus world knowledgebase engineering process, knowledge engineering should decide ⇒ Whether knowledgebase need to be able to choose actions or it is required to answer questions only about the contents of the environment ?

2) Assemble the relevant knowledge :

i. This is the process of knowledge acquisition.

ii. At this stage knowledge is gathered.

iii. This step helps to understand how the domain actually works.

iv. This process is carried out by an expert in the underlying domain. (either knowledge base engineer himself or knowledgebase engineer can take expert's help).

v. For example : In Wumpus world, the set of rules, the related actions are understood in this phase of knowledgebase engineering process.

vi. In real world domain it is difficult to study and gather knowledge as environment is complex.

For example : In VLSI design one needs to consider stray capacitances and skin effects.

3) Decide vocabulary of predicates, functions, and constants :

i. This step move towards formal representation of knowledge assembled in step 2.

ii. Domain-level concepts are translated into logic level names. The process of translation will involve lots of questions about domain like, what is this ?, What it likes ?, What it do ?, How it changes states etc.

iii. The translated version of problem domain knowledge will contain predicates, functions, constants, terms. All these things together form the vocabulary of problem-domain.

iv. Vocabulary of problem-domain is called as **Ontology**. The ontology determines what kinds of things exists, but does not determine specific properties and relationship about them.

4) Encode knowledge about the domain :

i. In this step knowledge engineer formally list down the axioms for all the vocabulary terms. It is noting of the meaning of the terms, enabling the expert to check the content.

ii. If some terms are found missing or irrelevant then the mistakes are corrected using step 3 again.

iii. If additional terms are found they are added to ontology. Invalid terms are removed from ontology. Thus the ontology is updated.

5) Encode a description of the specific problem instance : -

i. If ontology is well defined encoding the description becomes easy.

ii. This step involves writing simple atomic sentences about instances of various concepts that are part of ontology.

For example : For a logical agent various problem instances can be supplied by its sensors.

For disembodied knowledge base the instances can be generated by expected input.

6) Fire queries to inference procedure and get outputs : -

i. In this step the trials (demos) of querying the knowledge base are conducted.

ii. The inference procedure is applied on axioms and problem specific facts to get the results.

7) Debug the knowledgebase :

i. If step 6 succeds, its the reward of the first 5 steps.

ii. But if one is getting unexpected results of queries then knowledge base is debugged for errors.

iii. There are various reasons why inferencing can fail. The axioms can be missing, axioms can be weak to infer related query, axioms is wrong, are some of the reasons for failure.

For example :

The axiom $\forall$ x Numoflegs (x, 4) $\Rightarrow$ Mammal (x)

The above statement is false for reptiles. Here some other necessary axioms are missing.

iv. Step 6 and 7 will be carried out repeatedly to make 100 % error free knowledge base.

6.3.3 Knowledge Engineering in First Order Logic for the Electronic Circuits Domain

• Electronic circuit domain example - One-bit full adder

1. Identify the task : Does the circuit actually add properly ? (circuit verification)
2. Assemble the relevant knowledge :

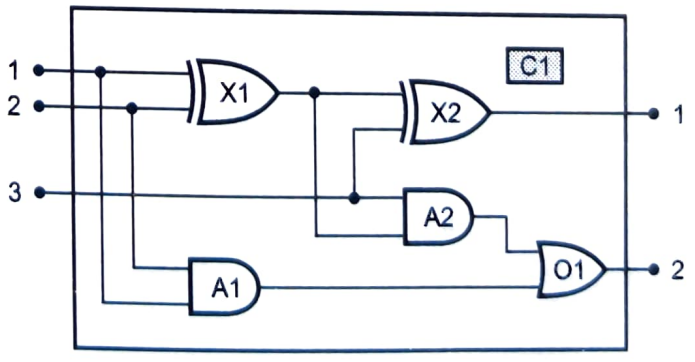


Fig. 6.3.1 A digital circuit C1 designed for one bit full adder

It is composed of wires and gates, types of gates (AND, OR, XOR, NOT). Irrelevant - Size, Shape, Color, Cost of gates.

3. Decide on a vocabulary / encoding :

   Alternatives -

   Type (X_1) = XOR

   Type $(X_1$, XOR)

   XOR (X_1)

   ....

4. Encode general knowledge of the domain :

   $\forall$ t_1 , t_2 Connected $(t_1, t_2) \Rightarrow$ Signal (t_1) = Signal (t_2)

   $\forall$ t Signal (t) = 1 $\vee$ Signal (t) = 0 $1 \neq 0$

   $\forall$ t_1, t_2 Connected $(t_1, t_2) \Rightarrow$ Connected (t_2, t_1)

   $\forall$ g Type (g) = OR $\Rightarrow$ (Signal (out (1, g)) = 1 $\Leftrightarrow$ $\exists$ n Signal (In (n, g)) = 1)

   $\forall$ g Type (g) = AND $\Rightarrow$ (Signal (out (1, g)) = 0 $\Leftrightarrow$ $\exists$n Signal (In (n, g)) = 0)

   $\forall$ g Type (g) = XOR $\Rightarrow$ (Signal (out (1, g)) =1 $\Leftrightarrow$ Signal (In (1, g)) $\neq$ Signal (In(2, g))

   $\forall$ g Type (g) = NOT $\Rightarrow$ Signal (out (1, g)) $\neq$ Signal (In (1, g)).

5. Encode the specific problem instance :

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

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

   Type (O_1) = OR.

   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))

6. Pose the queries to the inference procedure.

   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, C – 1)) =

$i_1 \wedge$ 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

7. Debug the knowledge base -
   May be you have omitted assertions like $1 \neq 0$, etc. ?

6.4 Inference in First Order Logic

GTU : Summer-18, Winter-18

We have seen how to make sound and complete inference for Propositional logic. These results can be extended to obtain algorithms that can answer any question (if it has answer) stated in first order logic.

6.4.1 Inferencing in First Order Logic

The semantics of first order logic provides a basis for a formal theory of logical inferences. These new derived inferences are correct in the sense that they are consistent with all previous interpretations of the original set of expressions.

The inference rules provide a computationally feasible way to determine when an expression, a component of an interpretation, logically follows for that interpretation.

The concept "**logically follows**" provides basis for proofs of the soundness and correctness of inference rules.

An interpretation that makes a sentence true is said to satisfy that sentence. An interpretation that satisfies every member of a set of expressions is said to satisfy the set.

An expression X logically follows from a set of first order logic expressions S if every interpretation that satisfies S also satisfies X. The function of logical inference is to produce new sentence that logically follow a given set of first order logic sentences.

The inference rule is essentially a mechanical means of producing new first order logic sentence from other sentences.

When every sentence X produced by an inference rule operating on a set S of logical expressions, logically follows from S, the inference rule is said to be **sound**.

If the inference rule is able to produce every expression that logically follows from S, then it is said to be **complete**.

6.4.2 Definition of Various Terms used in Inference Theory

1. **Satisfy, Model :**

- For a first order logic expression X and an interpretation I,
  - If X has a value of T under I and a particular variable assignment, then I is said to satisfy X.
  - If I satisfies X for all variable assignments, then I is a model of X.

2. **Inconsistent** :

- For a first order logic expression X and an interpretation I,
  - X is satisfiable if and only if there exist an interpretation and variable assignment that satisfy it; otherwise it is unsatisfiable.
  - A set of expressions is satisfiable if and only if there exist an interpretation and variable assignment that satisfy every element.

  If a set of expressions is not satisfiable, it is said to be inconsistent.

3. **Valid** :

- For a first order logic expression X and an interpretation I, If X has a value T for all possible interpretations, X is said to be valid.

4. **Proof Procedure** :

- A proof procedure is combination of an inference rule and an algorithm for applying that rule to a set of logical expressions to generate new sentences.

6.4.3 Inference Rules

1) **Modus ponens** :

If the sentence **P** and **P -> Q** are known to be true, then modus ponens lets us infer **Q**.

For example : If we have statment, " If it is raining then the ground will be wet" and "It is raining". If P denotes " It is raining" and Q is "The ground is wet" then the first expression becomes $P \rightarrow Q$. Because it is indeed now raining (P is true), our set of axioms becomes,

$\{ P \rightarrow Q$

$P \}$

Through an application of modus ponens, the fact that "The ground is wet" (Q) may be added to the set of true expressions.

- **The generalized modus ponens :**

For atomic sentences P_i, P_i', and q, where there is a substitution Q such that SUBST $(\theta, P_i') =$ SUBST (θ, P_i'),

For all i,

$$\frac{P_1', P_2', \ldots . P_n', (P_1 \wedge P_2 \wedge \ldots . \wedge P_n \Rightarrow q)}{SUBST(\theta, q)}$$

There are n+1 premises to this rule : - The 'n' atomic sentences p_i' and the one implication. The conclusion is the result applying the substitution θ to the consequent q.

P_1' is - Princess (Cindrella) P_1 is - Princess (x)

P_2' is - Kind (y) P_2 is - Kind (x)

θ is {x/Cindrella, y/cindrella} q is GoodHearted (x)

SUBST (θ, q) is, GoodHearted (cinderella).

2) **Modus tollens** :

Under the inference rules modus tollens, if $\mathbf{P \rightarrow Q}$ is known to be true and **Q** is known to be false, we can infer $\neg$ **P**.

For example : Given that,

i) $\underbrace{\text{Engine Starts} \wedge \neg \text{ Flat Tire}}_{P} \Rightarrow \underbrace{\text{Car OK}}_{Q}$

ii) $\underbrace{\neg \text{ Car OK}}_{Q}$

Then by Modus tollens,

$\underbrace{\neg (\text{Engine Starts} \wedge \neg \text{ Flat Tire})}_{\neg P}$ which is equivalence to $\neg$ [Engine Starts $\vee$ FlatTire]

3) **And elimination** :

And elimination allows us to infer the truth of either of the conjuncts from the truth of a conjunctive sentence. For instance, $\mathbf{P \wedge Q}$ is true. Lets us conclude that **P** and **Q** are true.

4) **And introduction** :

And introduction lets us infer the truth of a conjunction from the truth of its conjuncts. For instance, if **P** and **Q** are true, then $\mathbf{P \wedge Q}$ is true.

5) **Inference rules for quantifiers** :

1. Universal Instantiation (UI) :

This rule states that, we can infer any sentence obtained by substituting a ground term (a term without variable) for the variable.

Formally, this rule is described with the concept of substitution.

Substitution :

SUBST (θ, S) denotes the result of applying the substitution θ to the sentence S.

With the substitution we can write UI rule as follows,

$$\frac{\forall \ v \ \ S}{\text{SUBST}(\{v / g\}, S)}$$

For any variable V, ground term g.

Note *UI can be applied many times to produce many different consequences.*

For example :

All beautiful princess are GoodHearted.

$\forall$ x Princess (x) $\wedge$ Beautiful (x) $\Rightarrow$ GoodHearted (x)

From above axioms we can infer following permissible sentences : -

i. Princess (Seeta) $\wedge$ Beautiful (Seeta) $\Rightarrow$ GoodHearted (Seeta)

ii. Princess (Indumati) $\wedge$ Beautiful (Indumati) $\Rightarrow$ GoodHearted (Indumati)

iii. Princess (Mother (Seeta)) $\wedge$ Beautiful (Mother (Seeta)) $\Rightarrow$ Good Hearted (Mother (Seeta))

2. Existential Instantiation (EI)

This rule states that, "For any sentence S, variable V and constant symbol k that does not appear else where in the KB, following statement holds,

$$\frac{\exists\ V, S}{SUBST(\{V\ /\ k\}, S)}$$

Basically the existential statement say that, there is some object satisfying a condition. The instantiation process is just giving a name to this object. [It should be noted that this name must not already belong to another object].

For example :

$\exists$ x Crown (x) $\wedge$ OnHead (x, Cindrella)

We can infer sentence,

Crown (H) $\wedge$ On Head (H, Cindrella) as long as H does not appear else where in the knowledge base.

Note *Existential Instatiation can be applied once and then existentially quantified sentence can be discarded.*

For example :

If we add sentence,

Arrested (Police, Thief) then we do not need,

$\exists$ x Arrested (x, Thief)

6.4.4 Reducing First Order Logic Inferences to Propositional Inferences

1) In first order logic, we can use rules of quantified sentences to infer non-quantified sentences.
2) Using this rules we can reduce first order logic inferences to Propositional logic inferences.

3) The technique : -

- Existential quantified sentence can be replaced by one instantiations. (i.e one non-quantified sentence).
- Universal instantiation sentence can be replaced by all possible instantiations.

  For example :

  $\forall$ x Princess (x) $\wedge$ Beautiful (x) $\Rightarrow$ GoodHearted (x).

  We can apply universal instantiation to this sentence, using all ground terms in knowledgebase.

For example :

We can have ground terms,

Princess (Cindrella)

Beautiful (Cindrella),

then from universal instantiation we get,

GoodHearted (Cindrella)

We can add the sentence to knowledge base and discard universal quantified sentence. This technique is called as Propositionalization. Once we get all terms in Propositional logic, we can apply all algorithms of Propositional logic to these sentences.

4) Every first order logic knowledge base and first order logic query can be Propositionalized in such a way that entailment is preserved. It is a complete decision procedure.

5) Critical Problem : When a knowledge base contains functional symbol a set of possible ground term substitution is infinite.

   For example :

   'Mother' symbol can be infinitely nested like,

   Mother (Mother (Mother (Seeta)))

6) For first order logic, if we are deriving inference, the algorithm for inferring is surely going to output true for every entailed sentence. But there is no algorithm which outputs false to every non-entailed sentence. This is termed as **semidecidable property** of first order logic for entailment.

6.4.5 Concept of Lifting

1) Lifting means giving additional facilities (that is generating new modified version with upgradation).
2) Generalized Modus Ponen is lifted version of Modus Ponen (A rule in Propositional logic).

3) Lifting of the rule is done from Propositional logic version to first order logic version.

4) Similar to Modus ponen we are going to study lifted upgrated versions [from Propositional logic to first order logic] of unification, forward chaining, Backward Chaining and resolution procedures.

6.4.6 Unification

1) It is the process of finding substitutions for lifted inference rules, which can make different logical expression to look similar (identical).

2) Unification is an procedure for determining substitutions needed to make two first order logic expressions match.

3) Unification is important component of all first order logic inference algorithms.

4) The unification algorithm takes two sentences and returns a unifier for them, if one exists.

The algorithm statement is as follows : -

Unify (p, q) = θ where,

SUBST (θ, p) = SUBST (θ, q).

For example :

Suppose we have query-whom does Manmohan meets ? i.e first order logic query is Meets (Manmohan, x)

Some answers to this query can be found by searching all the sentences in knowledge base that unify with Meets (Manmohan, x).

Following are results with 4 different sentences that might be in knowledge base.

i. Unify (Meets (Manmohan, x), Meets (Manmohan, Sonia)) = { x/Sonia}

ii. Unify (Meets (Manmohan, x), Meets (y, Sharad)) = { x/Sharad, y/Manmohan}

iii. Unify (Meets (Manmohan, x), Meets (y/Friend (y))) = { y/Manmohan, x/Friend (Manmohan)}

iv. Unify (Meet (Manmohan, x), Meets (x, Mayawati)) = Fail

The last unification failed.

Note that, Meets (x, Mayawati) means everybody meet Mayawati. It was very straight forward that Manmohan meets Mayawati. But problem arised due to clash between variable name x, which appeared in both sentences.

Solution to variable name-clash :

We can rename variable in one of the sentences where it is clashed. This is called as **Standardizing apart** one of the two sentences.

For example :

In our Meets sentence,

Meets (x, Mayawati) we rename x as z.

∴ Unify (Meets (Manmohan, x), Meets (z, Mayawati)) = { x/Mayawati, z/Manmohan}

Algorithm Unify

```
Function unify (E1, E2) ;
begin
Case
both E1 and E2 are constants or the empty list :   // Recursion stops
    If E1 = E2 then return {}
    else return FAIL ;
E1 is a variable :
    If E1 occurs in E2 then return FAIL
    else return {E2/E1} ;
E2 is a variable :
    if E2 occurs in E1 then return FAIL
    else return {E1/E2}
either E1 or E2 are empty then return FAIL // the lists are of different sizes
 Otherwise : // both E1 and E2 are lists.
    begin
        HE1 : = First element of E1 ;
        HE2 : = First element of E2 ;
        SUBS1 : = Unify (HE1, HE2) ;
        if SUBS1 : = FAIL then return FAIL ;
        TE1 : = apply (SUB1, rest of E1) ;
        TE2 : = apply (SUB1, rest of E2) ;
        SUBS2 : = Unify (TE1, TE2) ;
        if SUBS2 = FAIL then return FAIL ;
        else return composition (SUBS1, SUBS2)
     end
    end                    // end case
  end
```

- **Most General Unifier : (MGU) :**

1) We are finding substitution that makes two arguments look the same.
2) There can be more than one substitutions that can make two arguments look the same.

   In our example of Meets sentence

   Unify (Meets (Manmohan, x), Meets (y,z))

   will return

   { y/Manmohan x/z}

   OR { y/Manmohan, x/Manmohan, z/Manmohan}
3) From above example we can conclude that

   First unifier gives Meets (Manmohan, z)

   Second unifier gives Meets (Manmohan, Manmohan)

   The second unifier is obtained from first by an additional substitution,

   { z/Manmohan}

   Here, we say that first unifier is more general than the second because it places fewer restrictions on the values of the variables.
4) For every such unifiable pair of expressions, there is a single most general unifier which is unique upto renaming of variables.

   In our example { y/Manmohan, x/z}
5) General steps to find Most General Unifier (MGU) and problems associated in finding MGU.

   i) The algorithm recursively explore the two expressions simultaneously "side by side", building up a unifier along the way. It fails at a stage when two corresponding points in the structure do not match.

   ii) The algorithm has one expensive step, (in terms of multiple checks). When matching a variable for a complex term, one must check whether the variable itself occurs inside the term. If it does occurs then the match fails because no consistent unifier can be constructed. This **"Occur check"** make algorithm expensive in terms of time. The time complexity becomes equivalent to quadratic in the size of the expressions being unified.

   iii) Because of huge time complexity some systems drops the **"Occur check"**. But these systems then can become unsound inference systems. Therefore many systems use different procedures for finding MGU, than the one we have discussed here.

Unification example :

A Tree diagram representing unification algorithm procedure :

- Initial steps in the unification of (Parents X (Father X) (Mother bill)) and (Parents bill (Father bill) Y).

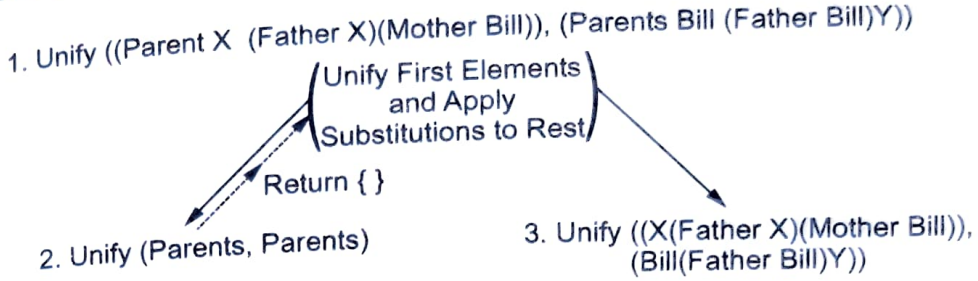


Fig. 6.4.1 Unification Process I

- Further steps in unification of (Parents X (Father X) (Mother bill) and (Parents bill (Father bill) Y)

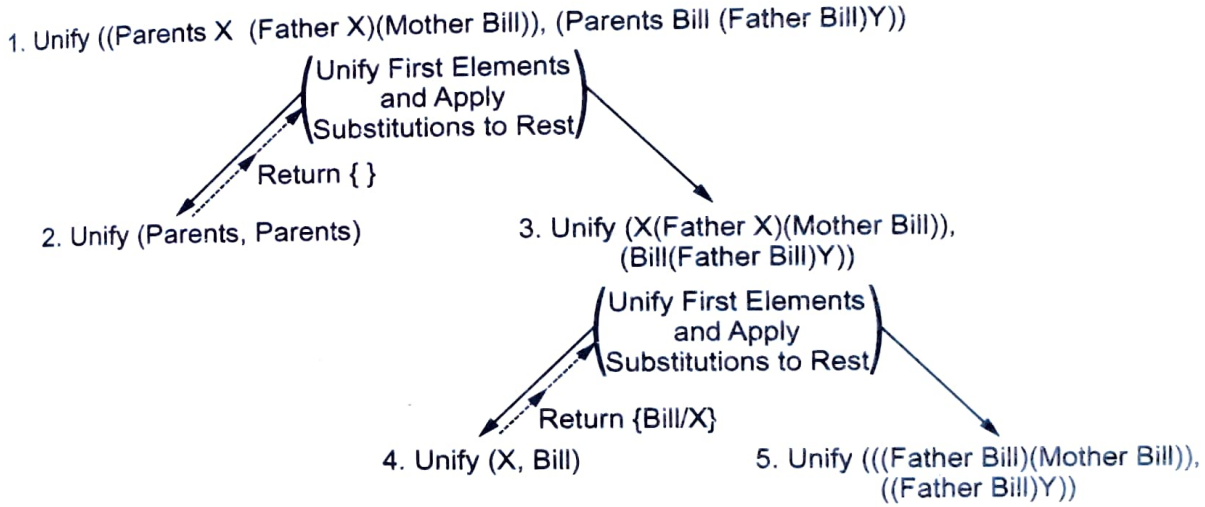


Fig. 6.4.2 Unification Process II

- Final trace of the unification of (parents X(father X) (mother bill)) and (parents bill (father bill) Y). (See Fig. 6.4.3 on next page.)

6.4.7 Storage and Retrieval of Data from Knowledge Base

1) Knowledge base has basic two functions for storing and then fetching the data for use.
   i. The TELL function is used to store data in knowledge base.
   ii. The FETCH function is used to retrieve data from knowledge base.

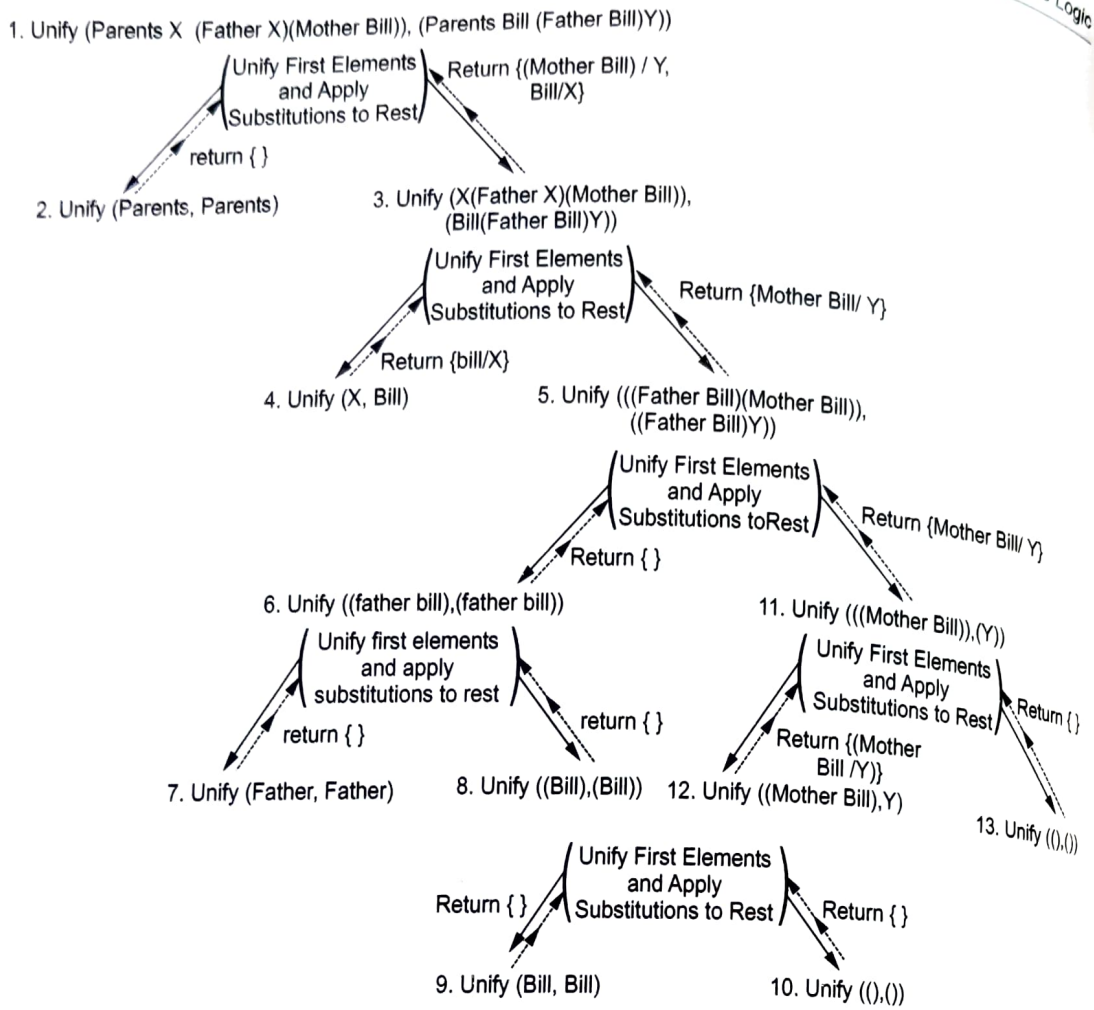


Fig. 6.4.3 Unification Process III

2) Internally TELL function is implemented as STORE(S) primitive (basic functionality), which stores a sentence S into the knowledge base.

3) Internally ASK function is implemented as FETCH(q) primitive (basic functionality), which returns all unifiers such that the query 'q' unifies with some sentence in knowledge base.

Knowledge base implemented as a long list

1) The simplest way to implement STORE and FETCH is to keep all the facts in the knowledge base in one long list.

2) The given query 'q' will then call UNIFY (q, s) for every sentence S in the list.

3) Such a process is inefficient in terms of time taken because it will unify each and every statement of knowledge base (which may unify unnecessary sentences).

4) In such case FETCH can be made efficient by unifying those sentences only which will have a chance to be unified.

For example :

There is no point in trying to unify

Meets (Manmohan, x) with the sensence,

Father (Ram, Manmohan)

5) We can avoid such unification by indexing the facts in the knowledge base.

6) A simple indexing called predicate indexing puts all the "Meets" facts in one bucket (stored collection) and all the 'Father' facts in another bucket. These buckets (stored collection) can be stored in hash table for fast access.

7) Predicate indexing is useful when there are many predicate symbols and there are few clauses for each symbol.

If there are many clauses for single symbol then the bucket of this symbol will be too large and again the search will be inefficient (more time incurred).

8) We can solve large buckets problem with the help of multiple indexing.

We can index fact both by predicate and by first or second argument of the fact by using composite (combined) key for hash table.

Then we can simply construct the key from the query and retrieve exactly those facts that unify with the query.

If we index the fact with first argument combined with predicate then it can be stored under multiple index keys.

(P which will answer various queries, unified by this predicate).

9) Given a sentence to be stored in a knowledge base it is possible to construct indices for all queries that unify with it.

For example :

The fact TEACHES (Radhakrishnan, Dipu)

Will have following sets of queries

a. TEACHES (Radhakrishnan, Dipu)

Does RadhaKrishnan teaches Dipu ?

b. TEACHES (x, Dipu)

Who teaches Dipu ?

c. TEACHES (Radhakrishnan, y)

Whom does Radhakrishnan teach ?

d. TEACHES (x, y)

Who teaches whom ?

10) The subsumption lattice :

i. The above set of queries are said to form a collection tree called as **Subsumption lattice.**

ii. Properties of subsumption lattice :

a. The child of any node in the lattice is obtained from its parent by single substitution.

b. The highest common descendent of any two nodes is the result of applying their most general unifier.

c. The portion of the lattice, above any ground fact can be constructed systematically as shown in following figure,

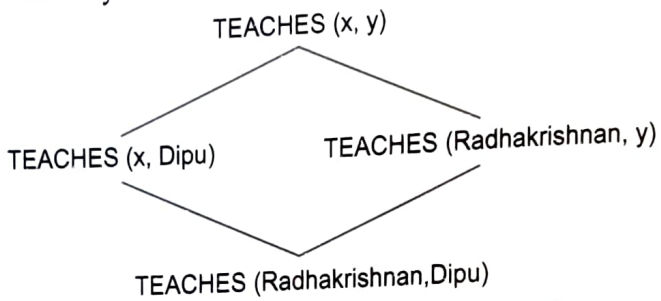


Fig. 6.4.4 Portion of subsumption lattice

A subsumption lattice whose lowest is the sentence,

TEACHES (Radhakrishnan, Dipu).

d. A sentence with repeated constants has a slightly different lattice as shown in following figure.

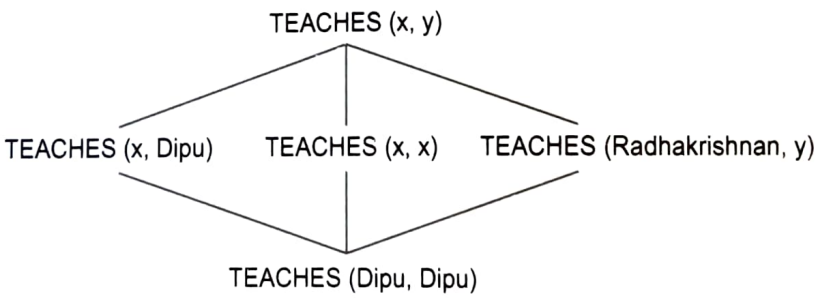


Fig. 6.4.5 Portion of subsumption lattice with repeated constants

e. Function symbols and variables in the sentence will have different lattice structure.

f. The lattice representation work well when it contains small number of nodes.

g. For a predicate with n arguments, the lattice contains O (2^n) nodes.

6.4.8 First Order Definite Clauses

1) They are disjunctions of literals of which exactly one is positive.
2) A definite clause is either atomic sentence or is an implication whose antecedants (left hand side clause) is a conjunction of positive literals and consequent (Right hand side clause) is a single positive literal.

   For example :

   Princess (x) $\wedge$ Beautiful (x) $\Rightarrow$ GoodHearted (x)

   Princess (x)

   Beautiful (x)
3) First order logic literals can include variables, in which case those variables are assumed to be universally quantified.
4) Generally, universal quantifiers are omitted while writing definite clauses.
5) Definite clauses are suitable formal forms to used with generalized modus ponen.
6) Not every knowledge base can be converted into a set of definite clause, because of **single-positive-literal condition.**
7) Set of FOL definite clauses with no function symbols is called as **Datalog knowledge base.**
8) The absence of function symbols in Datalog Knowledge base helps to make inferencing easy.

6.4.9 Inferencing Algorithm for First Order Logic

6.4.9.1 Forward Chaining (Lifted Forward Chaining)

1) Forward chaining is applied to first order logic definite clauses.
2) Definite clauses such as, situations $\Rightarrow$ Response are especially useful for systems that make inference in response to newly arrived information. Forward chaining can be more efficient than resolution theorem proving if systems are represented in definite clauses.
3) Starting from known fact, it triggers all the rules whose premises are satisfied, and add their conclusions to known facts.
4) The process repeats until query is answered (assuming there is only one answer expected) or no new facts are added (that is query remains unanswered).
5) Fact is **not considered** new if it is simply renamed version of existing fact. One sentence is renaming of another sentence if they are identical except for the names of the variable.

For example :

Likes (x, Rose) and

Likes (y, Rose)

are renaming of each other because they differ only in variable names. Both means the same "Everyone likes Rose".

6) In the algorithm a situation comes when no more new inferences are possible. At this stage, knowledge base is called as **Fixed Point** of the inference process.

First Order Logic Fixed Point (KnowledgeBases) can include universaly quantified atomic sentences.

7) Performance measurement of forward chaining : -

i. It is sound. Every inference is just an application of generalized Modus ponen which is sound.

ii. It is complete for definite clause knowledge base. It answers query if knowledge base entails it. One can prove completeness of Datalog in simple steps.

iii. If K- is the maximum arity (number of arguments) of any predicate, P- the number of predicates and n - number of constant symbols then there can be maximum (P*n)K distinct ground facts. After this, algorithm will reach fixed point.

8) Definite clauses with function symbols can generate infinitely many new facts. So extra care should be taken here to avoid infinite knowledge base growth.

9) If the query has no answer the algorithm may fail to terminate in some cases.

Forward chaining procedure :

Function FOL-FC-ASK (KB, α) returns a substitution or false

inputs : KB, the knowledge base, a set of first-order definite clauses. α, the query, at atomic sentence.

Local variables : New, the new sentences inferred on each iteration.

repeat until new is empty

new $\leftarrow$ {}

For each sentence r in KB do

{ $P_1 \wedge \ldots\ldots \wedge P_n \Rightarrow q$) $\leftarrow$ STANDARDIZE - APART (r) / * Resolve name clash */

For each θ such that SUBST (θ, $P_i \wedge \ldots\ldots \wedge P_n$) = SUBST ($\theta$, $P_i \wedge \ldots\ldots P_n$)

For some $P'_i \ldots\ldots P'_n$ in KB

$q' \leftarrow$ SUBST (θ, q)
if q' is not a renaming of some sentence already in KB or new
then do
add q' to new
$\phi \leftarrow$ UNIFY (q', α)
if ϕ is not fail then return ϕ add new to KB.
return false.

Forward chaining example :

Consider the following problem :

"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 Colone West, who is American".

We need to prove "West is Criminal".

The first order definite clause for above problem domain are as follows : -

1) ".... it is a crime for an American to sell weapons to hostile nations" : -

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

2) "Nono...has some missiles". The sentence $\exists$ x Owns (Nono, x) $\wedge$ Missile (x) is transformed into two definite clauses by existential elimination, introducing a new constant M_1 : $\subset$

   Owns (Nono, M_1)

   Missile (M_1)

3) "All of its missiles were sold to it by Colonel West" : $\subset$

   Missile (x) $\wedge$ Owns (Nono, x) $\Rightarrow$ Sells (West, x, Nono)

4) "We will also need to know that missiles are weapons :

   Missile (x) $\Rightarrow$ Weapon (x)

5) We must know that an enemy of America counts as "hostile" : $\subset$

   Enemy (x, America) $\Rightarrow$ Hostile (x).

6) "West, who is American .... " :$\subset$ American (West).

7) "The country Nono, an enemy of America ...." : $\subset$ Enemy (Nono, America).

Note

- *In the proof tree initial facts appear at the bottom level.*
- *Facts inferred on the first iteration appear in the middle level.*
- *Facts inferred on the second interation appear at the top level.*

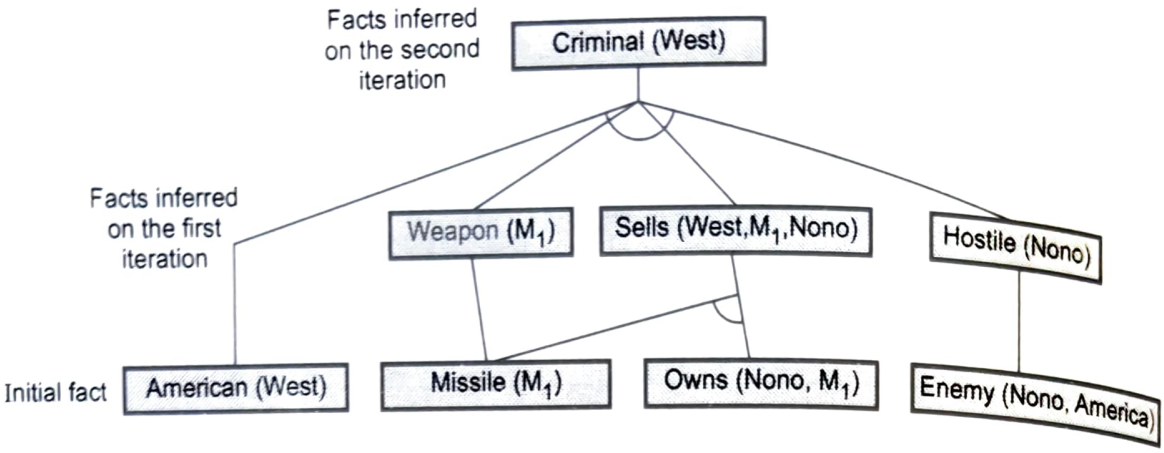


Fig. 6.4.6 A proof tree generated by forward chaining algorithm on crime example

Efficiency of Forward Chaining :

Source of complexity for first order logic forward chaining - ASK are as follows,

1) Inner loop searches all unifiers therefore Pattern matching is expensive.
2) Every rule is tested again in each iteration.
3) Algorithm may produce many facts not relevant for the goal.

1) Matching rules with known facts :

i. To apply Missile (x) ⇒ Weapon (x), find all facts which unify with Missile (x).

↳ This can be done in constant time using indices.

ii. To apply Missile (x) ∧ Owns (Nono, x) ⇒ Sells (West, x, Nono),

We can either,

↳ First find all objects Nono owns and then test if the objects are missiles or

First find all missiles and then test if they are owned by Nono.

↳ This is the problem of conjugate order.

a) Choose and order which minimizes overall costs (depends on knowledgebase).

b) Use heuristics, example, "most constraint variable" aka "Minimum Remaining Value (MRV)".

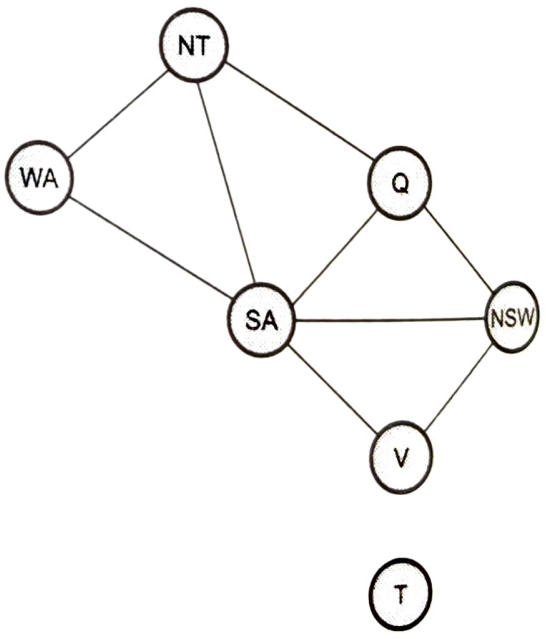


Fig. 6.4.7 Constrait graph for colouring the map

c) Pattern matching has strong relation to CSPs !

d) Every conjunct is a constraint for the contained variables.

iii. Hard matching example :

Consider the constraint graph for colouring the map [Refer chapter 3 for detail description]

- Every CSP with finite domain can be expressed as follows : -
- As a single definite clause : -

  Diff (wa, nt) $\wedge$ Diff (wa, sa) $\wedge$ Diff (nt, q)

  $\wedge$ Diff (nt, sa) $\wedge$ Diff (q, nsw) $\wedge$ Diff (q, sa)

  $\wedge$ Diff (nsw, v) $\wedge$ Diff (nsw, sa) $\wedge$ Diff (v, sa) $\Rightarrow$ Colorable ( )
- With the according facts : -

  Diff (Red, Blue) Diff (Red, Green)

  Diff (Green, Red) Diff (Green, Blue)

  Diff (Blue, Red) Diff (Blue, Green)

- Colorable ( ) is inferred iff the CSP has a solution,
- CSPs include 3SAT as a special case, hence matching is NP - hard.
- NP completeness of forward chaining in its inner loop put into following aspects perspective : -

a) Most rules in real world knowledge bases are small and simple in contrast to CSP formulation.

i) In database world :

1. Often there are limits for rule sizes and predicate arity.
2. Inference complexity is just dependent on number of facts.

b) Generate subclasses of rules which are efficient.

1. Each Datalog clause can be seen as CSP.
2. Solve the CSP, if it is a tree, it is solvable in linear time.
3. Same can be done for the rules, for example, delete SA in prior example. That is,

Diff (wa, nt) $\wedge$ Diff (nt, q) $\wedge$ Diff (q, nsw) $\wedge$ Diff (nsw, v) $\Rightarrow$ Colorable ( )

c) It avoids redundant rule matches.

2) Incremental forward chaining :

1. Simple first order logic - forward chaining - ASK would repetitively and redundantly match rules,

example,

Missile (x) ⇒ Weapon (x) during second iteration.

Observation :

No need to match rule on iteration k if a premise wasn't added on iteration (k–1)!

a. Match each rule whose premise contains a newly added positive literal.

b. Match rule its premises contain a fact P_i' which unifies with a fact P_i' derived during k – 1.

c. The algorithm

```
The algorithm for every iteration k
 for every rule r
   for each Pi in premises (r)
                for each Pi derived during k–1
          if unify (Pi, P'i) ⇒ match (r)
```

2. Database indexing allows O(1) retrival of known facts.

   Example :

   Query Missile (x) retrieves Missile (M_1)

3. Redundancy can be avoided if partial derivations are buffered :

   Rete - Algorithm : It uses a data propagation network which propagates variable bindings.

   Every node is literal from premises.

4. Rete and successors were basis for production systems like, XCON (DEC) hardware configuration and OPS-5, a general language or for cognitive architectures like ACT (Anderson, 1983) or SOAR (Laired et al., 1987)

3) Irrelevant Facts :

1. Deduction of facts is not required for a given goal (similar to forward chaining in propositional logic.)
2. Solution : -

   (i) Use of subset of rules (see propositional logic).

   (ii) From deductive database research : Use a magic set

- Only consider rules with a given variable binding : For e.g. if goal is Criminal (West), then the rule that concludes Criminal (x) can be written with extra conjuct that constraints the value of x. That is,

  Magic (x) ∧ American (x) ∧ Weapon (y) ∧ Sells (x, y, z) ∧ Hostile (z) ⇒ Criminal (x)

6.4.9.2 Backward Chaining (Lifted Backward Chaining)

1) It use generalized Modus ponen backwards to prove query q, and work backwards.

   - It checks if q is known already.

   - Otherwise it proves by backward chaining all premises of a rule concluding q.

2) The list of goals can be thought of as a "stack" waiting to be worked on; if all of them can be satisfied, then the current branch of the proof succeeds. The algorithm takes the first goal in the list and finds every clause in the knowledge base whose positive literal, or head, unifies with the goal. Each such clause creates a new recursive call in which the premise, or body, of the clause is added to the goal stack.

3) The algorithm uses composition of substitutions. COMPOSE (θ_1, θ_2) is the substitution whose effect is identical to the effect of applying each substitution in turn. That is,

   SUBST (COMPOSE (θ_1, θ_2), P) = SUBST (θ_2, SUBST (θ_1, P)).

   In the algorithm, the current variable bindings, which are stored in θ, are composed with the bindings resulting from unifying the goal with the clause head, giving a new set of current bindings for the recursive cell.

The backward chaining Procedure :

Function FOL-BC-ASK (KB, goals, θ) return a set of substitutions.

Inputs : KB, a knowledgebase goals, a list of conjucts forming a query (θ already applied)

θ, the current substitution, initially the empty substitution { }

Local variables : answers , a set of substitutions, initially empty.

if goals is empty then return {θ}

$q' \leftarrow$ SUBST (θ, FIRST (goals))

For each sentence r in knowledge base where STANDARDIZE - APART

(r) = ($P_1 \wedge ..... \wedge P_n \Rightarrow q$)

and $\theta' \leftarrow$ UNIFY (q, q') succeeds new-goals $\leftarrow$ [P_1, ..., P_n/REST (goals)]

answers $\leftarrow$ FOL-BC-ASK (KB, new-goals, COMPOSE (θ', θ)) U answers

return answers.

Properties of backward chaining :

1) It uses depth-first search for proof. It's memory requirement is linear in size of proof.

2) It is incomplete due to infinite loops. It should check if new subgoal is already an goal stack.

3) It is inefficient due to repeated subgoals (success and failure). It should cache previous results and check for new subgoal.

4) We can have two versions of it. Find any solution, OR Find all solutions.

5) It is used for logic programming : Prolog

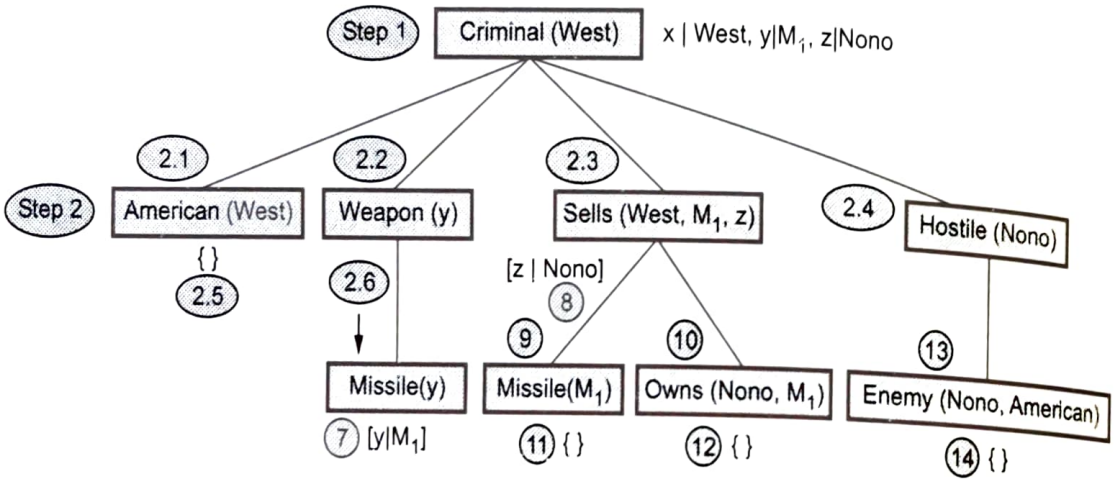


Fig. 6.4.8 Proof tree generated by backward chaining to prove that 'West is criminal'

- **Backward chaining example :**

Note *The tree should be read depth first, left to right.*

To prove criminal (west), we have to prove the four conjucts below it.

Some of the required conjucts are in knowledgebase and some conjucts are needed to be proved.

Bindings of for each successful unification are shown next to the corresponding subgoal.

Once one subgoal in a conjuction suceeds, its substitution is applied to subsequent subgoals. Thus, by the time FOL-BC-ASK gets to the last conjuct, originally. Hostile (z), z is already bound to Nono.

6.4.9.3 Forward Chaining Vs Backward Chaining

1) Forward chaining is data driven.
   - It is automatic unconscious processing.
   - Example - Object reorganition, routine decisions.
   - It may do lots of work that is irrelevant to the goal.

2) Backward chaining is goal driven.
- It is appropriate for problem solving.
- Example : Where are my keys ?,
  How do I get into a PhD programme ?
- Complexity of backward chaining can be much less than linear in size of knowledgebase.

6.4.10 Resolution

It is procedure of inferencing the given statement based on available data. The resolution procedure we are going to study is the lifted version of resolution procedure in propositional logic.

In first order logic, for resolution, it requires the sentences in Conjunctive Normal Form (CNF). That is, a conjunction of clauses, where each clauses is a disjunction of literals. Literals can contain variables, which are assumed to be universally quantified.

- General format of conjunctive normal form with exactly K literals per clause is
  $(l_{1,1} \vee .... \vee l_{1,K}) \wedge .... \wedge (l_{n,1} \vee ..... \vee l_{n,K})$
- Generally there can be any number of literals in a single clause.
  For example :
  $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$
- CNF statement with quantifiers -
  $[\forall x\ A(x) \vee \forall y\ B(y)] \wedge [\exists z\ C(z)]$
- Consider another example :
  $\forall x$ **American (x)** $\wedge$ **Weapon (y)** $\wedge$ **Sells (x, y, z)** $\wedge$ **Hostile (z)** $\Rightarrow$ **Criminal (x)**
  The above sentence in CNF becomes, $\neg$ **American (x)** $\vee \neg$ **Weapon (y)** $\vee$
  $\neg$ **Sells (x, y, z)** $\vee \neg$ **Hostile (z)** $\vee$ **Criminal (x)**
- Note that,
  CNF sentence will be unsatisfiable only when original sentences is unsatisfiable
  Therefore we will have resolution proofs by contradiction on the CNF sentences.

• The general steps for converting first order logic sentences to conjunctive normal form

1. $(\forall x)(P(x)) \Rightarrow ((\forall y)(P(y) \Rightarrow P(F(x,y))) \wedge \neg(\forall y)(Q(x,y) \Rightarrow P(y)))$(like $A \Rightarrow B \wedge C$)

2. Eliminate $\Rightarrow$

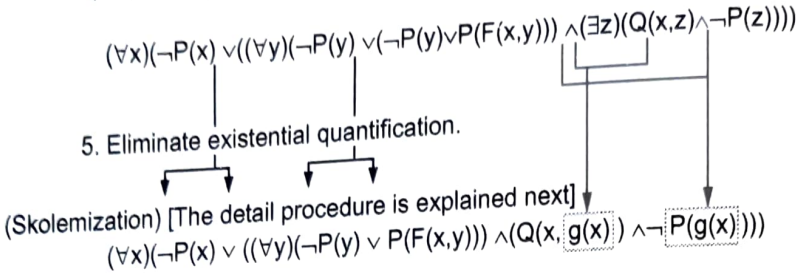
$(\forall x)(\neg P(x) \vee ((\forall x)(\neg P(y) \vee P(F(x,y))) \wedge \neg(\forall y)(\neg Q(x,y) \vee P(y)))$

3. Reduce scope of negation :

$(\forall x)\ (\neg P(x) \vee ((\forall y)\ (\neg P(y) \vee P\ (F(x, y))) \wedge (\exists y)\ (Q(x, y) \wedge \neg P(y))))$

4. Standardize variables :

$$(\forall x)\ (\neg P(x) \vee ((\forall y)\ (\neg P(y) \vee P(F(x, y))) \wedge (\exists z)\ (Q(x, z) \wedge \neg P(z))))$$



6. Drop universal quantification symbols :

$$(\neg P(x) \vee ((\neg P(y) \vee P(F(x, y))) \wedge (Q(x, g(x)) \wedge \neg P(g(x)))))$$

7. Convert to conjunction of disjunctions :

$$(\neg P(x) \vee \neg P(y) \vee P\ (F(x, y))) \wedge (\neg P(x) \vee Q(x, g(x))) \wedge (\neg P(x) \vee \neg P(g(x))).$$

$$(\neg P(x) \vee \neg P(y) \vee P\ (F(x, y))) \wedge (\neg P(x) \vee Q(x, g(x))) \wedge (\neg P(x) \vee \neg P(g(x)))P$$

8. Create separate clauses :

$$\neg P(x) \vee \neg P(y) \vee P(F(x, y))$$

$$\neg P(x) \vee Q\ (x, g(x))$$

$$\neg P(x) \vee \neg P(g(x)).$$

9. Standardize variables :

$$\neg P(x) \vee \neg P(y) \vee P(F(x, y))$$

$$\neg P(z) \vee Q\ (z, g(z))$$

$$\neg P(w) \vee \neg P(g(w)).$$

Skolemize :

1) Skolemization is the process of removing existential quantifier by elimination.
2) This methods converts a sentence with existential quantifier into a sentence without existential quantifier such that the first sentence is satisfiable if and only if the second is.
3) To eliminate an existential quantifier, replace each occurrence of its variable by a skolem function whose arguments are the variables of universal quantifier whose scope includes the scope of the existential quantifier being eliminated.
4) If the existential quantifier being eliminated is not within the scope of any universal quantifier, the skolem function has no arguments that is, it is a constant.

Skolemization example :

1. $\forall x \exists y$ (Person (x) $\wedge$ Person (y)) $\Rightarrow$ Loves (x, y)

   is converted to

   $\forall x$ (Person (x) $\wedge$ Person f(x) $\Rightarrow$ Loves (x, f(x))

   Where f(x) specifies the person that x Loves.

2. The sentence "Everyone has a brain" is represented as

   $\forall x$ Person (x) $\Rightarrow \exists y$ Brain (y) $\vee$ Has (x, y)

   If we simply substitute the constant B for the existentially qualified variable y, we get a sentence that says "Everyone has the same brain", not what we want to say!

   $\forall x$ Person (x) $\Rightarrow$ Brain (B) $\wedge$ Has (x, B)

   Using the Skolem function B(x) to represent a function that denotes the object that is x's brain, the correct skolemization of our original sentence is,

   $\forall$ x Person (x) $\Rightarrow$ Brain (B(x)) $\wedge$ Has (x, B(x)).

3. $\exists$ x Rich (x) becomes Rich (G1) where G1 is a new "Skolem constant"

   $\exists$ k $\frac{d}{dy}(K^y) = K^y$ becomes $\frac{d}{dy}(e^y) = e^y$ more tricky when $\exists$ is inside $\forall$

Example : "Everyone has a heart".

$\forall x$ Person (x) $\Rightarrow \exists y$ Heart (y) $\wedge$ Has (x, y)

Incorrect : $\forall x$ Person (x) $\Rightarrow$ Heart (H1) $\wedge$ Has (x, H1).

Correct : $\forall x$ Person (x) $\Rightarrow$ Heart (H(x)) $\wedge$ Has (x, H(x))

[If x has a y we can infer that y exists. However, its existence is contingent on x, thus y is a function of x as H(x)].

Where H is new symbol ("Skolem" function)

Skolem function arguments : All **enclosing** universally quantified variables.

Detail example :

Converting first order logic (predicate logic) sentences in to CNF :

1) The conversion procedure is almost the same as Propositional logic with the major difference arises where we need to eliminate existential quantifier.

2) We will consider sentence,

   "Everyone who loves all animals is loved by someone".

   The first order logic representation for above sentence is,

   $\forall$ x [$\forall$y Animal (y) $\Rightarrow$ Loves (x, y)] $\Rightarrow$ [$\exists$y Loves (y, x)]

3) The steps to convert above first order logic sentence into CNF are as follows,

Consider following sentence,

$\forall x\ [\forall y\ \text{Animal}\ (y) \Rightarrow \text{Loves}\ (x, y)] \Rightarrow [\exists y\ \text{Loves}\ (y, x)]$

The steps are as follows :

1. Elimination implications :

$\forall x\ [\neg\ \forall y \neg\ \text{Animal}\ (y) \vee \text{Loves}\ (x, y)] \vee [\exists y\ \text{Loves}\ (y, x)]$

2. Move $\neg$ inwards :

In addition to the usual rules for negated connectives, we need rules for negated quantifier. Thus we have,

i. $\neg\ \forall x\ P$ becomes $\exists x \neg\ P$

ii. $\neg\ \exists x\ P$ becomes $\forall x \neg\ P$.

Our sentence goes through the following transformations :

i. $\forall x\ [\ \exists y \neg\ (\neg\ \text{Animal}\ (y) \vee \text{Loves}\ (x, y))] \vee [\exists y\ \text{Loves}\ (y, x)]$.

ii. $\forall x\ [\ \exists y \neg \neg\ \text{Animal}\ (y) \wedge \neg\ \text{Loves}\ (x, y)] \vee [\exists y\ \text{Loves}\ (y, x)]$.

iii. $\forall x\ [\ \exists y\ \text{Animal}\ (y) \wedge \neg\ \text{Loves}\ (x, y)] \vee [\exists y\ \text{Loves}\ (y, x)]$.

Note *How a universal quantifier ($\forall y$) in the premise of the implication has become an existential quantifier. The sentence now reads "Either there is some animal that x doesn't love, or (if this not the case) someone loves x," Clearly, the meaning of the original sentence has been preserved.*

3. Standardize Apart Variables (Remove name clash)

For sentences like $(\forall x\ P(x)) \vee (\exists x Q(x))$ which use the same variable name twice, change the name of the variables. This avoids confusion later when we drop the quantifier. Thus we have $\forall x\ [\exists y\ \text{Animal}\ (y) \wedge \neg\ \text{Loves}\ (x, y)] \vee [\ \exists\ z\ \text{Loves}\ (z, x)]$.

4. Apply this rule to our sample sentence, we obtain

$\forall x\ [\text{Animal}\ (A) \wedge \neg\ \text{Loves}\ (x, A)] \vee \text{Loves}\ (B, x)$.

which has the wrong meaning entirely : It says that everyone either fails to love a particular animal A or is loved by some particular entity B. In fact, our original sentence allows each person to fail to love a different animal or to be loved by a different person. Thus we want the skolem entities to depend on x.

$\forall x\ [\text{Animal}\ (F(x)) \wedge \neg\ \text{Loves}\ (x, F(x))] \vee \text{Loves}\ (G(x), x)$.

Here F and G are Skolem functions.

5. Drop-universal quantifiers :

   At this point, all remaining variables must be universally quantified. More over, the sentence is equivalent to one in which all the universal quantifier have been moved to the left. We can therefore drop the universal quantifier.

   [Animal (F(x)) $\wedge \neg$ Loves (x, F(x))] $\vee$ Loves (G(x), x).

6. Distribute $\wedge$ over $\vee$:

   [Animal (F(x)) $\vee$ Loves (G(x), x)] $\wedge$ [$\neg$ Loves (x, F(x))] $\vee$ Loves (G(x), x)]

   This step may also require flattening out nested conjunctions and disjunctions.

   The sentence is now in CNF and consists of two clauses.

- **The resolution inference rule :**

1) The resolution rule for first order clauses is simply a lifted version of the propositional resolution rule.
2) Two clauses, which are assumed to be standardized apart so that they share no variables, can be resolved if they contain complementary literals.
3) First order literals are complementary if one unifies with the negation of the other.
4) The resolution inference rule is

$$\frac{l_1, V \ldots V\, l_k, \qquad m_1 V \ldots V\, m_n}{\text{SUBST}(\theta, l_1, V \ldots V\, l_{i-1}\, V,\ l_{i+1} V \ldots V\, l_k\, V_m, V \ldots V_{mj-1} V_{mj+1} V \ldots V_{m_n})}$$

   where UNIFY (l_i, $\neg$ mj) = θ for example,

   we can resolve the two clauses,

   [Animal (F(x)) $\vee$ Loves (G(x), x)] and [$\neg$ Loves (u, v) $\vee \neg$ kills (u, v)]

   by eliminating the complementary literals Loves (G(x), x) and $\neg$ Loves (u, v), with unifier.

   θ = {u/G(x), v/x}, to produce the resolvent clause.

   [Animal (F(x)) $\vee \neg$ kills (G(x), x)]

5) The rule we have just given is the binary resolution rule, because it resolves exactly two literals. The binary resolution rule by itself does not yield a complete inference procedure.
6) The full resolution rule resolves subsets of literals in each clause that are unifiable.
7) Another approach is to extend factoring. Factoring means the removal of redundant literals in first order case. Propositional factoring reduces two literals

to one if they are identical; first order factoring reduces two literals to one if they are unifiable.

Statement proving by resolution :

Resolution proves that KB $\models \alpha$ by proving KB $\wedge \neg \alpha$ unsatisfiable, i.e, by deriving the empty clause. It is a proof by contradiction.

Following are steps taken in resolution proof,

1) Convert all problem statements to first order logic.
2) Convert first order logic statements into conjunctive normal form.
3) Assert the negation of the goal.
4) Resolve clauses together until FALSE is derived.

Algorithm always chose to resolve with a clause whose positive literal unifies with the leftmost literal of the 'current' clause. This is exactly what happens in backward chaining. (See Fig. 6.4.9 on next page.)

Resolution example 1 :

Problem statement in English,

1. "Everyone who loves all animals is loved by someone".
2. "Anyone who kills an animal is loved by no one."
3. "Jack loves all animals."
4. "Either Jack or Curiosity killed the cat, who is named Tuna."

Statement to prove

"Did Curiosity kill the cat ?"

I) Expressing English Sentences

1) Express the knowledge in FOL.
2) The original sentences.
3) Some background knowledge, and
4) The negated goal G in first-order logic.
5) The original sentences in first order logic.

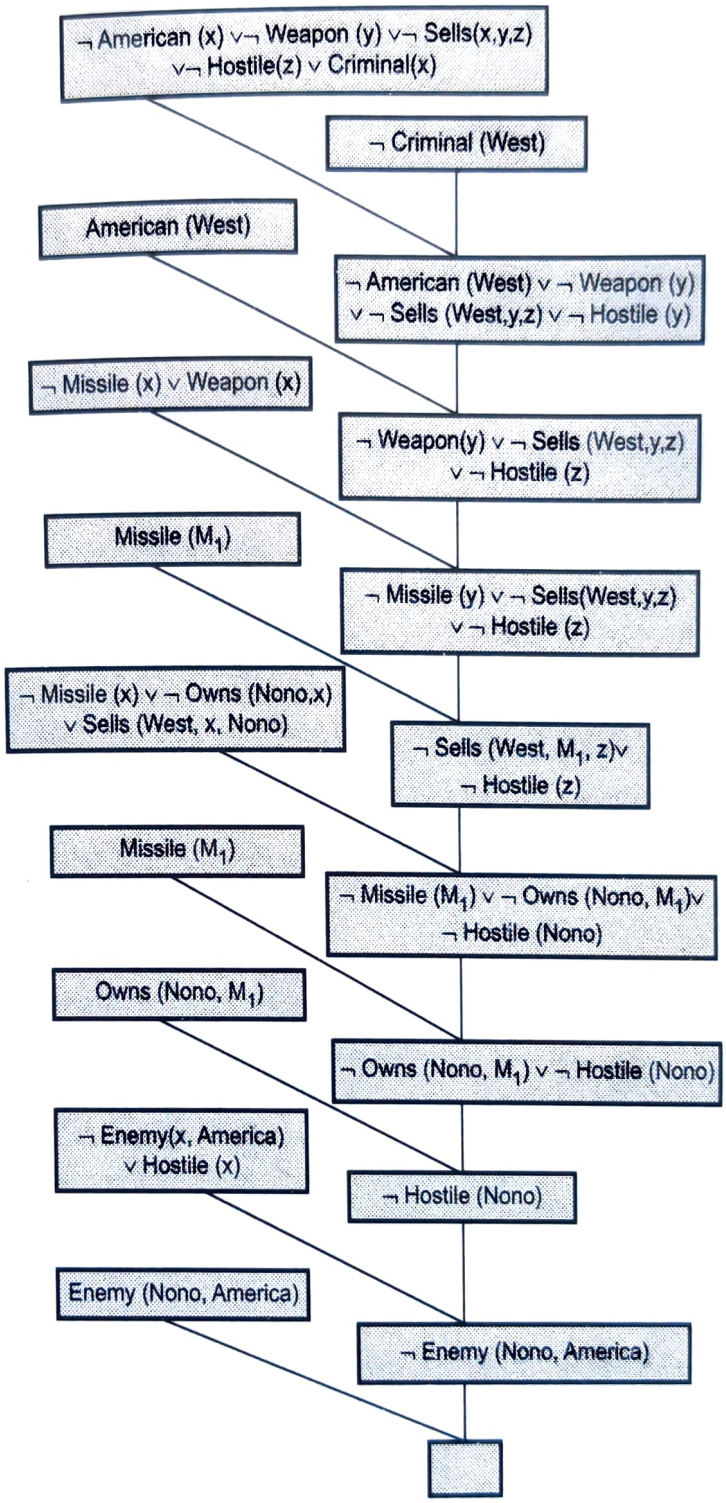


Fig. 6.4.9 A resolution proof that "West is Criminal"

English statement 1 :

A. $\forall x [\forall y \text{ Animal } (y) \Rightarrow \text{Loves } (x, y)] \Rightarrow [\exists y \text{ Loves } (y, x)]$.

English statement 2

B. $\forall x [\exists y \text{ Animal } (y) \wedge \text{kills } (x, y)] \Rightarrow [\forall z \neg \text{ Loves } (z, x)]$.

English statement 3

C. $\forall x \text{ Animal } (x) \Rightarrow \text{Loves (Jack, x)}$

English statement 4

D. Kills (Jack, Tuna) $\vee$ Kills (Curiosity, Tuna).

2) Some background knowledge

E. Cat (Tuna)

F. $\forall x \text{ Cat } (x) \Rightarrow \text{Animal } (x)$

3) Negated goal G in first order logic

$\neg$ G. $\neg$ Kills (Curiosity, Tuna).

II) Apply the conversion procedure to convert each sentence to conjunctive form :

A1. Animal (F(x)) $\vee$ Loves (G(X), x)

A2. $\neg$ Loves (x, F(x)) $\vee$ Loves (G(x), x)

B. $\neg$ Animal (y) $\vee$ $\neg$ Kills (x, y) $\vee$ $\neg$ Loves (z, x)

C. $\neg$ Animal (x) $\vee$ Loves (Jack, x)

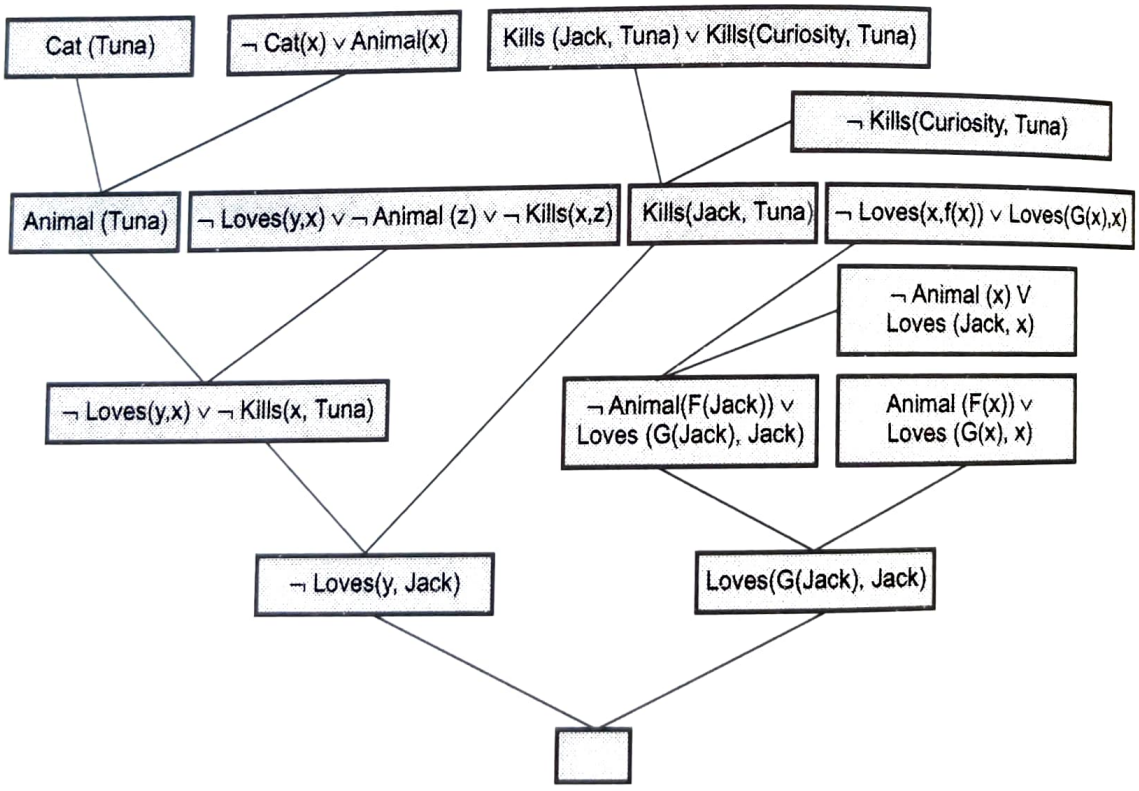


Fig. 6.4.10 Resolution Tree : "Did Curiosity kill the cat ?"

D. Kills (Jack, Tuna) ∨ Kills (Curiosity, Tuna)

E. Cat (Tuna)

F. ¬ Cat (x) ∨ Animal (x)

G. ¬ Kills (Curiosity, Tuna)

- The resolution proof for the statement "Curiosity killed the cat"

Note that, the use of factoring in the derivation of the clause Loves (G(Jack), Jack).

Resolution example 2 :

Anyone passing his/her history exams and winning the lottery is happy. But anyone who studies or is lucky can pass all his/her exams. Nimu did not study. Nimu is lucky. Anyone who is lucky wins the lottery.

To prove : Is Nimu happy ?

Step 1 : Convert to first order logic.

1) Anyone passing his history exams and winning the lottery is happy.

   ∀ x Pass (x, History) ∧ Win (x, Lottery) Happy (x)

2) But anyone who studies or is lucky can pass all his exams.

   ∀ x ∀ y Study (x) ∨ Lucky (x) Pass (x, y)

3) Nimu did not study, but Nimu is lucky.

   ¬ Study (Nimu) ∧ Lucky (Nimu)

4) Anyone who is Lucky wins the lottery.

   ∀ x Lucky (x) Win (x, Lottery)

Step 2 : Convert to CNF

i) Eliminate implications :

1. ∀ x ¬ (Pass (x, History) ∧ Win (x, Lottery)) ∨ Happy (x).
2. ∀ x ∀ y ¬ (Study (x) ∨ Lucky (x)) ∨ Pass (x, y)
3. ¬ Study (Nimu) ∧ Lucky (Nimu)
4. ∀ x ¬ Lucky (x) ∨ Win (x, Lottery)

Move ¬ inward

1. ∀ x ¬ Pass (x, History) ∨ ¬ Win (x, Lottery)) ∨ Happy (x).
2. ∀ x ∀ y (¬ Study (x) ∧ ¬ Lucky (x)) ∨ Pass (x, y).
3. ¬ Study (Nimu) ∧ Lucky (Nimu).
4. ∀ x ¬ Lucky (x) ∨ Win (x, Lottery).

ii) Standardize variables :

No action needed.

iii) Move quantifier left :

No action needed except drop quantifier.

iv) Skolemize :

No action needed.

v) Distribute $\wedge$ over $\vee$

1. $\neg$ Pass (x, History) $\vee \neg$ Win (x, Lottery)) $\vee$ Happy (x).
2. ($\neg$ Study (x) $\vee$ Pass (x, y)) $\wedge$ ($\neg$ Lucky (x) $\vee$ Pass (x, y)).
3. $\neg$ Study (Nimu) $\wedge$ Lucky (Nimu).
4. $\neg$ Lucky (x) $\vee$ Win (x, Lottery).

vi) Flatten nested conjunctions and disjunctions : **no action necessary.**

vii) State as a set of disjunction of literals

1. $\neg$ Pass (x, History) $\vee \neg$ Win (x, Lottery)) $\vee$ Happy (x).
2. a) $\neg$ Study (x) $\vee$ Pass (x, y)
   b) Lucky (x) $\vee$ Pass (x, y)
3. a) $\neg$ Study (Nimu)
   b) Lucky (Nimu)
4. $\neg$ Lucky (x) $\vee$ Win (x, Lottery).

viii) Standardize Variables apart

1. $\neg$ Pass (x1, History) $\vee \neg$ Win (x1, Lottery) $\vee$ Happy (x1).
2. a) $\neg$ Study (x2) $\vee$ Pass (x2, y1)
   b) $\neg$ Lucky (x3) $\vee$ Pass (x3, y2)
3. a) $\neg$ Study (Nimu)
   b) Lucky (Nimu)
4. $\neg$ Lucky (x4) $\vee$ Win (x4, Lottery)

Now, this is in conjunctive normal form (CNF).

Step 3 : Resolution proof procedure

- Assert negation of goal :
  - In this case the goal is to prove **Happy (Nimu).**
  - Add the negation of the goal clause to the knowledge base $\neg$ **Happy (Nimu).**
- Resolve clauses together until FALSE is derived.

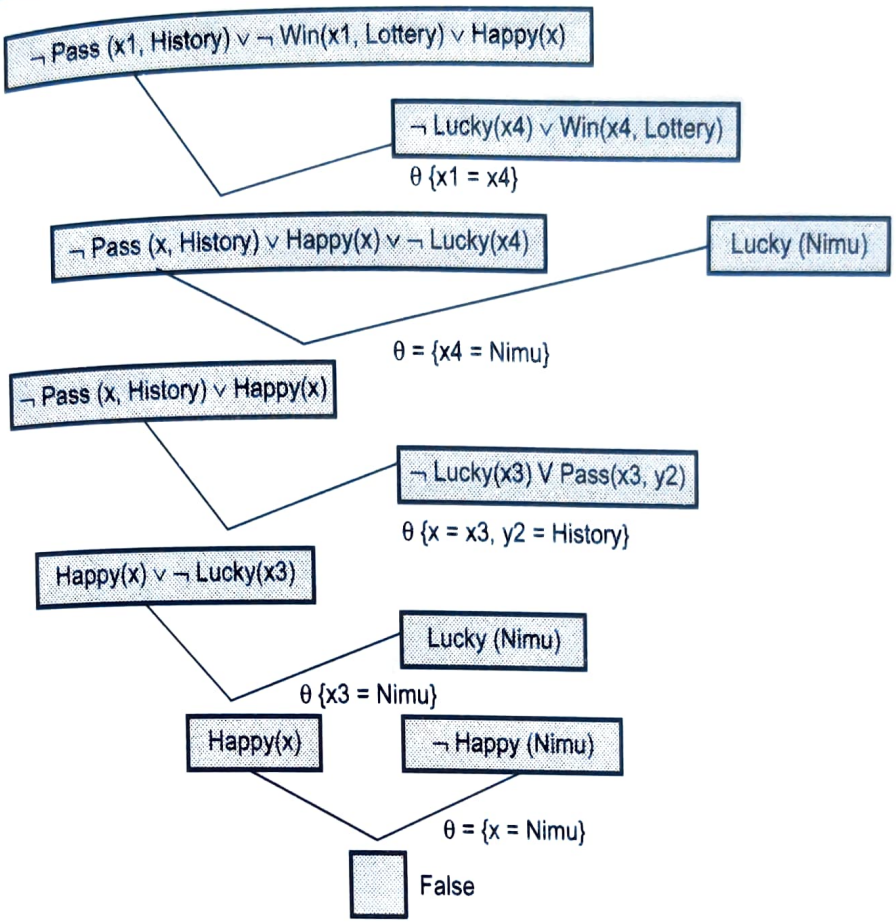


Fig. 6.4.11 Resolution proof tree for the goal-Happy(Nimu)

Resolution example 3 :

- Consider following knowledgebase

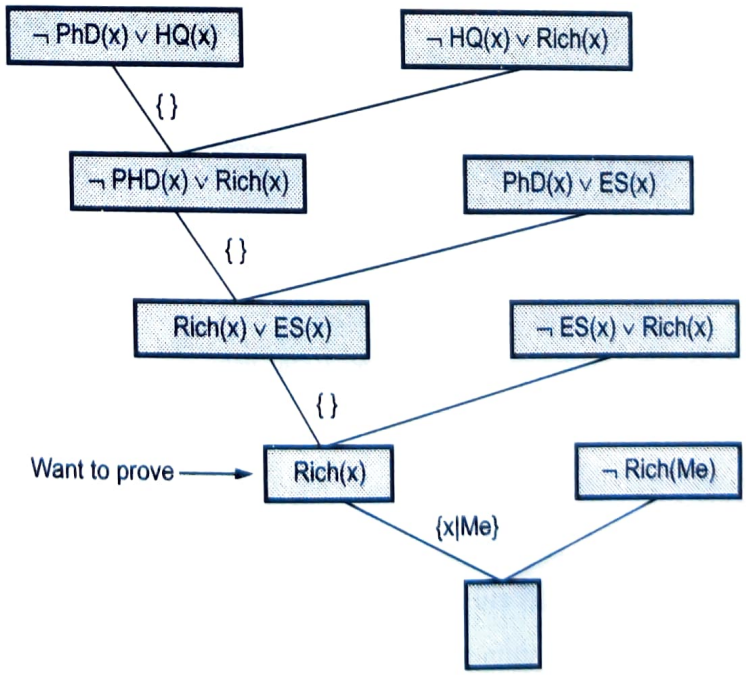


Fig. 6.4.12 Resolution proof tree for the goal Rich (me)

(i) ¬ PhD (x) ∨ HighlyQualified (x)

(ii) PhD (x) ∨ EarlyEarnings (x)

(iii) ¬ HighlyQualified (x) ∨ Rich (x)

(iv) ¬ Early Early Earnings (x) ∨ Rich (x)

- To prove **Rich (me), add ¬ Rich (me)** to the knowledge base

For resolution proof tree refer Fig. 6.4.12.

Resolution example 4 :

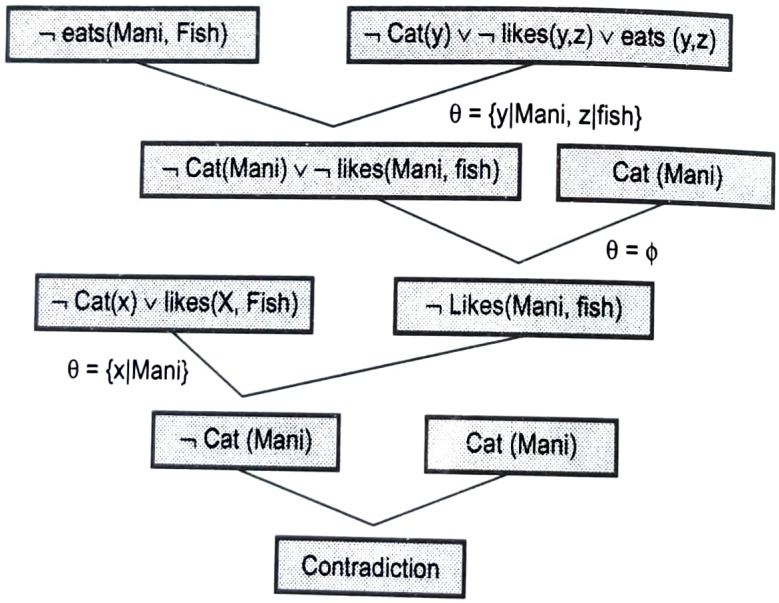
- Consider following knowledge base.

(i) Cats like fish ¬ Cat (x) ∨ likes (x, fish)

(ii) Cats eat everything they like ¬ Cat (y) ∨ ¬ likes (y, z) ∨ eats (y, z).

(iii) Mani is a cat Cat (Mani)

- To prove "Mani eats fish", **eats (Mano, fish) add ¬ eats (Mani, fish)** to the knowledge base.
- **Resolution proof tree**



Negation of goal : ¬ eats (Mani, fish)

Fig. 6.4.13 Resolution proof tree for the goal eats (mani, fish)

Resolution example 5

⇒ Consider following english sentences,

1) Everyone who loves all animals is loved by someone.

2) Anyone who kills an animal is loved by no one.

3) Jack loves all animals.

4) Either Jack or John killed the cat named Tuna.

⇒ Represent above sentences in FOL and prove that **"John killed the cat"**.

Step 1 : We need to express

i) The original sentences, ii) Some background knowledge, and

iii) The negated goal G in first order logic.

Step 1 works as follows,

i) The original sentences in first order Logic.

English statement 1 :

A. $\forall x [\forall y \text{ Animal } (y) \Rightarrow \text{Loves } (x, y)] \Rightarrow [\exists y \text{ Loves } (y, x)]$.

English statement 2 :

B. $\forall x [\exists y \text{ Animal } (y) \wedge \text{kills } (x, y)] \Rightarrow [\forall z \neg \text{Loves } (z, x)]$.

English statement 3 :

C. $\forall x \text{ Animal } (x) \Rightarrow \text{Loves (Jack, x)}$.

English statement 4 :

D. kills (Jack, Tuna) $\vee$ kills (John, Tuna)

ii) Some backward knowledge

E. Cat (Tuna)

F. $\forall x \text{ Cat } (x) \Rightarrow \text{Animal } (x)$

iii) Negated goal G in first order logic.

$\neg$ G. $\neg$ kills (John, Tuna)

Step 2 : Apply the conversion procedure to convert each sentence to conjunctive normal form :

A1. Animal (F(x)) $\vee$ Loves (G(x), x)

A2. $\neg$ Loves (x, F(x)) $\vee$ Loves (G(x), x)

B. $\neg$ Animal (y) $\vee \neg$ kills (x, y) $\vee \neg$ Loves (z, x)

C. $\neg$ Animal (x) $\vee$ Loves (Jack, x)

D. kills (Jack, Tuna) $\vee$ kills (John, Tuna)

E. Cat (Tuna)

F. $\neg$ Cat (x) $\vee$ Animal (x)

$\neg$ G. $\neg$ kills (John, Tuna)

Step 3 : The resolution proof tree for the statement "John killed the cat".

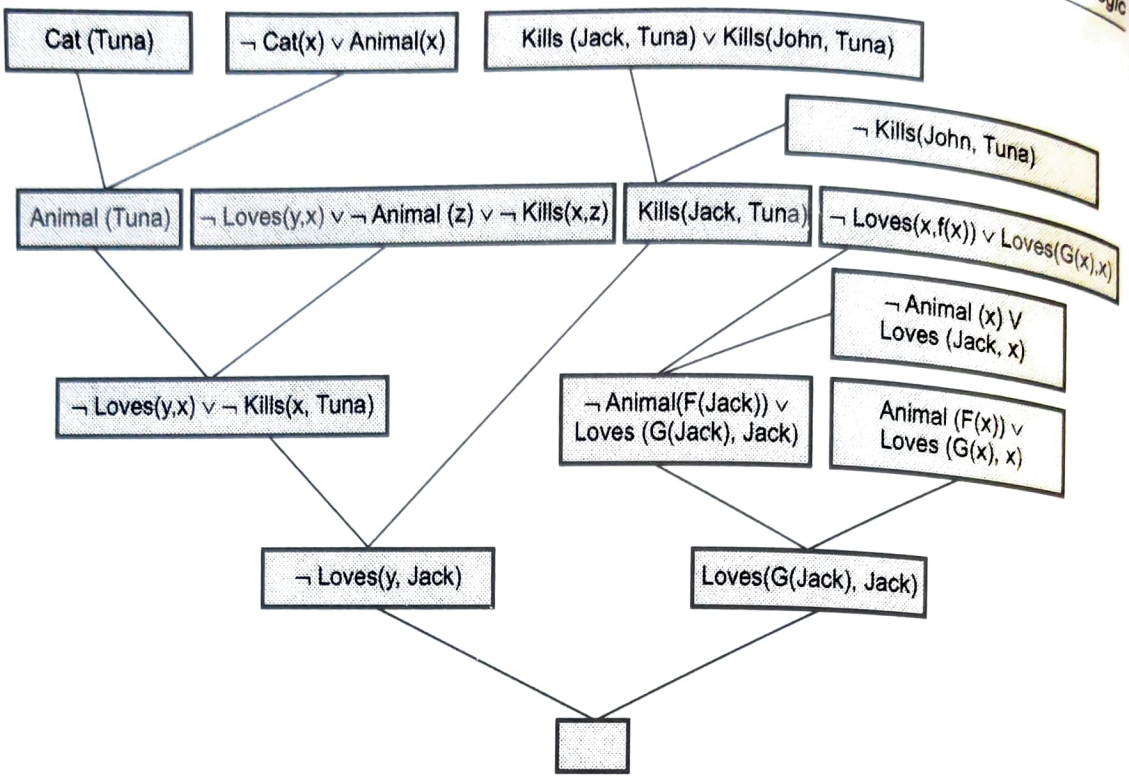


Fig. 6.4.14 Resolution Tree : "Did John kill the cat ?"

- **Completeness of resolution :**

1) Resolution is **refutation complete**, it means that if a set of sentences is unsatisfiable, then resolution will always be able to derive a contradiction.

2) Resolution cannot be used to generate all logical consequences of set of sentences, but it can be used to establish that a given sentence is entailed by the set of sentences. Hence, it can be used to find all answers to a given question, using the negated-goal method.

3) The statement of proof is,

   If S is an unsatisfiable set of clauses, then the application of a finite number of resolution steps to S will yield a contradiction.

A proof that proves, "Resolution proofs for entailment is complete". The basic structure of the proof :

Proof proceeds as follows :

1. First, we observe that if S is unsatisfiable, then there exists a particular set of ground instances of the clauses of S such that this set is also unsatisfiable (Herbrand's theorem).
2. We then appeal to the ground resolution theorem, which states that propositional resolution is complete for ground sentences.

3. We then use a lifting lemma - Lifting lemma states that for every resolution proof on a set of ground formulae, there exists a corresponding proof based on the non ground formulae that originate the ground ones.

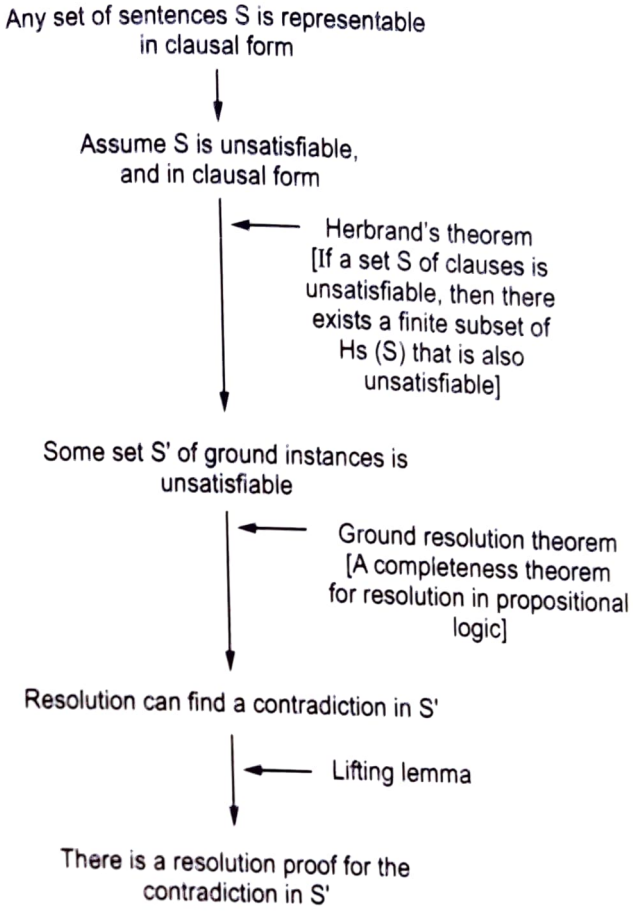


Fig. 6.4.15 Structure of a completeness proof for resolution

- **Resolution strategies :**

1) **Unit preference :**
   - Clauses with just one literal are prefered.
   - Unit resolution : Incomplete in general, complete for horn knowledgebase.

2) **Set of support :**
   - At least one of the clauses makes part from the set of support.
   - Complete if the remainder of the sentences are jointly satisfiable.
   - Using the negated query as set-of-support.

3) **Input resolution :**
   - At least one of the clauses makes part from the initial knowledgebase or the query.
   - Complete for horn, incomplete in the general case.

- Linear resolution : P and Q can be resolved if P is in the original knowledge base or P is an ancestor of Q in the proof tree; complete.

4) Subsumption :

- All sentences subsumed by others in the knowledgebase are eliminated.

Types of Resolution

There are various types of resolution are possible depending on the types and number of parent clauses as follows :

1) Binary resolution - Two clauses having complementary literals are combined as disjuncts to produce a single clause after deleting the complementary literals. For example

$$\neg P(x, a) \vee Q(x) \text{ and}$$

$$\neg Q(b) \vee R(x)$$

is just

$$\neg P(b, a) \vee R(b)$$

The substitution {b | x} was made in the two parent clauses to produce the complementary literals Q(b) and ¬ Q(b) which were then deleted from the disjunction of the two parent clauses.

2) Unit Resulting (UR) Resolution - A number of clauses are resolved simultaneously to produce a unit clause. All except one of the clauses are unit clauses, and that one clause has exactly one more literal than the total number of unit clauses. For example, resolving the set

$$\{ \neg \text{MARRIED}(x, y) \vee \neg \text{MOTHER}(x, z) \vee$$
$$\text{FATHER}(y, z), \text{MARRIED}(\text{sue}, \text{joe}),$$
$$\neg \text{FATHER}(\text{joe}, \text{bill}) \}$$

Where the substitution β = {sue | x, joe | y, bill | z} is used, results in the unit clause ¬ MOTHER (sue, bill).

3) Linear Resolution - When each resolved clause C_i is a parent to the clause C_{i+1} (i = 1, 2, ..., n – 1) the process is called linear resolution. For example, given a set S of clauses with $C_0 \subseteq S$, C_n is derived by a sequence of resolutions, C_0 with some clause B_0 to get C_1, then C_1 with some clause B_1 to get C_2 and so on until C_n has been derived.

4) Linear Input Resolution - If one of the parents in linear resolution is always from the original set of clauses (the Bi), then it is a linear input resolution. For example, given the set of clauses.

$S = \{ P \vee Q, \neg P \vee Q, P \vee \neg Q, \neg P \vee \neg Q \}$ let $C_0 = (P \vee Q)$. Choosing $B_0 = \neg P \vee Q$ from the set S and resolving this with C_0 we obtain the resolvent $Q = C_1$. B_1 must now be chosen from S and the resolvent of C_1 and B_1 becomes C_2 and so on.

5) Set-of-support resolution - Let S be an unsatisfiable set of clauses and T be a subset of S. Then T is a set-of-support for S if S-T is satisfiable. A set-of-support resolution is a resolution of two clauses not both from S – T.

This essentially means that given an unsatisfiable set $\{A_1, \ldots\ldots, A_K\}$, resolution should not be performed directly among the A_i.

Steps taken to speed up resolution

When the choice of the clauses to resolve together at each step is made in certain systematic ways, then the resolution procedure will find a contradiction if one exists. However, it may take a very long time. Strategies exist there so as to speed up the process considerably, as given below

1) Resolve only those pairs of clauses that contain complementary literals, since only such resolutions produce new clauses that are harder to satisfy than their parents.

2) Eliminate certain clauses as soon as they are generated so that they cannot participate in later resolutions, such as tautologies (which can never be unsatisfied) and that are subsumed by other clauses (i.e. they are easier to satisfy. For example, $P \vee Q$ is subsumed by P).

3) Whenever possible, resolve either with one of the clauses that is part of the statement which is one trying to refute or with a clause generated by a resolution with such a clause. This is called **set-of-support** strategy.

4) Whenever possible, resolve with clauses that have a single literal. Such resolutions generate new clauses with fewer literals than the larger of their parent clauses and thus are probably closer to the goal of a resolvent with zero terms. This method is called the unit-preference strategy.

For example consider following set of statements -

i) Marcus was a man.

ii) Marcus was a pompeian.

iii) All pompeians were Romans.

iv) Caesar was a Ruler.

v) All Romans were either loyal to Caesar or hated him.

vi) Everyone is loyal to someone.

vii) People only try to assassinate rulers they are not loyal to.

viii) Marcus tried to assassinate Caesar.

- Now, prove hate (Marcus, Caesar) i.e., Marcus hate Caesar.
- For this, first of all convert these facts into clause form as shown below

Axioms in Clause form

1. man (Marcus)
2. Pompeian (Marcus)
3. ¬ Pompeian (x_1) ∨ Roman (x_1)
4. ruler (Caesar)
5. ¬ Roman (x_2) ∨ loyalto $(x_2$, Caesar) ∨ hate$(x_2$, Caesar)
6. loyalto$(x_3$, f1$(x_3))$
7. ¬ man(x_4) ∧ ¬ ruler (y_1) ∨ ¬ tryassassinate (x_4, y_1) ∨ loyal to (x_4, y_1)
8. tryassassinate (Marcus, Caesar)

- For proving hate (Marcus, Caesar), negate the clause having

  ¬ hate (Marcus, Caesar) as the resulting clause.
- Now try to prove using steps given above. This procedure is shown in Fig. 6.4.16.

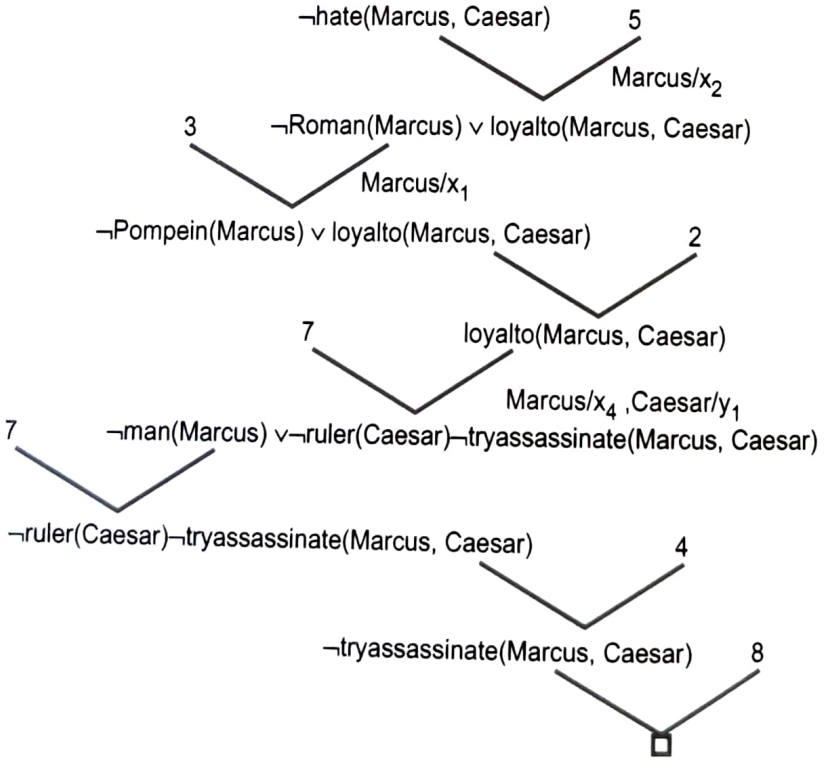


Fig. 6.4.16 A resolution proof hate (Marcus, Caesar)

- Start this proof with clause - hate (Marcus, Caesar) and consider, clause 5 as the second parent, telling x_2 = Marcus, the resolvent clause generated is

  ¬ Roman (Marcus) ∨ loyal to (Marcus, Caesar).

- Now, consider another contradictory clause i.e. clause 3 as parent with it and generate a resolvent clause.

  ¬ pompeian (Marcus) ∨ loyalto (Marcus, Caesar) telling Marcus = x_1.

- Further, clause 2 is considered and resulting clause is

  loyalto (Marcus, Caesar).

- Further, take 7^{th} clause as parent and resulting resolvent is

  ¬ Man (Marcus) ∨ ¬ Ruler (Caesar) ∨ ¬ tryassassinate (Marcus, Caesar)

  with putting X_4 = Marcus and y_1 = Caesar.

- Further, clause 1 is taken that generates resulting clause

  ¬ Ruler (Caesar) ∨ ¬ tryassassinate (Marcus, Caesar).

- Next, 4^{th} clause is taken that generates clause

  ¬ tryassassinate (Marcus, Caesar).

  Which with clause 8 result into an empty clause.

- Empty clause means that the clause which is initially taken totally contradicts and cannot be true any how. Which result that

  ¬ Hate (Marcus, Caesar) is false and its contradict

  i.e. Hate (Marcus, Caesar) is true.

6.4.11 Dealing with Equality

The unification algorithm described in the previous section will not unify pairs of sentences such as

i) isPriminister (M_Singh) and

ii) isPriminister (Manmohan_Singh), and rightly so, because syntactically, the constants M_Singh and Manmohan_Singh are quite different. We would not want our theorem proving agent to go around making assumptions about the equality of constants just based on their names, as this would make the system unsound. Surely we would want it to be able to unify the two isPriminister predicates, as this will allows it to do more resolution steps.

One way to get around the problem with equality is to add lots of extra clauses to our knowledge base, expressing every thing we wanted to say about equality. **These are known as equality axioms.** In particular, we would have to say that it is reflexive, symmetric and transitive :

$\forall x\ (x = x)$ [Reflexive]

$\forall x, y\ (x = y \rightarrow y = x)$ [Symmetric]

$\forall x, y, z\ (x = y \wedge y = z \rightarrow x = z)$ [Transitive]

This is not enough to enable the system to realize when it can unify two constants. We would have to put in additional statements about equality for each of the predicates in our knowledge base. So for example, if the predicate P of arity 1 was mentioned in the knowledge base, then we would have to add the following to the knowledge base :

$\forall x, y\ (x = y \rightarrow P(x) = P(y))$.

This kind of thing will have to be done for every predicate and function, if we are to be sure not to miss any opportunity for unification.

An alternative way to get a proving agent to use equality to its full is to employ **another inference rule called demodulation**. Like the resolution rule, this takes two sentences as input, but one of them must be an equality sentence relating two terms. Then, if, in the other sentence, there is a term T which can be unified with the left hand substituted for T. It's more obvious when written as following rule :

$$\frac{X = Y\ ,\ A\ [T]}{SUBST\ (\theta, A\ [y])...)}\ Unify\ (x, s) = \theta$$

We can also allow demodulation to work the other way, i.e. replace y with x, but as with rewritting we need to take care not to get stuck in any loops. Our presidential example would be a trivial case for the demodulation rule : -

$$\frac{Manmohan\_Singh = M\_Singh,\quad isPriminister\ (Manmohan\_\ Singh)}{isPriminister\ (M\_\ Singh)}$$

In this case, the unifying substitution is just the trivial identify substitution.

Most state of the art resolution theorem provers have resolution at their heart, but use a number of other inference rules, including demodulation, and its big brother **paramodulation**, which deals with cases where we only know that

$x = y \vee P(x)$

- Induction is done over many different structures.
- It allows reasoning about recursion or iteration.
  [It is useful for hardware or software verification].

6.4.12 Theorem Provers (Automated Reasoners)

1) They accept full first order logic sentences.
2) In most theorem provers, the synthetic form chosen for the sentences does not affect the result.

3) Application areas : Verification of software and hardware

4) In mathematics theorem provers have a high standing. Now a days they have come up with novel mathematical results.

5) For example, in 1996 a version of well known OTTER was the first to prove (eight days of computation) that the axioms proposed by Herbert Robbins in 1933 really define Boolean algebra.

6) Following are various types of Theorem provers.

i. Inductive Theorem Proving :

- Deduction is done by mathematical inducation.

$$\frac{P(o)\ \forall x \cdot (P(x) \rightarrow P(x+1))}{\forall x \cdot P(x)}$$

ii. Interactive theorem proving :

- In this theorem proving it is necessary to interact with humans in order to prove theorems of any difficulty.
- These are like Mathematician's assistant. Let a theorem prover do simple tasks while Mathematician develop a theory (example : Buchberger's Theorem)
- It is a Guided theorem proving. Users follows and guides computer proof attempt.
- It needs visualization tools for proof trees.

iii. Higher order theorem proving

- It provides deduction in higher order logics.
- It allows more natural and succinct statements.
- Higher order theorem prover are used for verification tasks [Example - Verification of cryptographic protocols].
- It uses induction and interactive control.

6.5 Monotonic and Non-Monotonic Reasoning

GTU : Winter-18,19, Summer-18,19,20

There are various reasoning approaches used to solve problems posed by uncertain, fuzzy and often changing knowledge in the problem world.

6.5.1 Monotonic Reasoning

1) It is the reasoning in which the axioms and for the rules of interference are extended to make it possible to reason with consistent and complete information.
2) The only way it can change is that the new facts can be added as they become available.

3) If these new facts are consistent with all the other facts that have already been asserted, then nothing will ever be retracted from the set of facts that are known to be true. This property is called monotonicity, and a reasoning based on this property is known as Monotonic reasoning.

4) It works with information that is consistent and complete with respect to the domain of interest.

5) In other words, all the facts that are necessary to solve a problem are present in the system or can be derived from those that are by conventional rules of first-order logic.

6) It is a method of reasoning under consistent, complete, unchanging and certain facts. It was implicity assumed that a sufficient amount of reliable knowledge was available with which to deduce confident conclusions.

7) Thus a monotonic reasoning system cannot work effectively in real life environments because -

   i) Information available is always incomplete.

   ii) As process goes by, situations change and so are the solutions.

   iii) Default assumptions are made in order to reduce the search time and for quick arrival of solutions.

8) Logic based systems are monotonic in nature, i.e., if a proposition is made which is true, it remains true under all circumstances. But in real life, all statements made do not necessarily mean that they are correct under all circumstances.

9) Whenever one make a statement, one do not make it in a ad hoc fashion. The statement is made by manipulating a set of beliefs. Experts predict, diagnose and perform majority of their mental activity by relying on their beliefs. It is possible that during the course of action, events may take place which can either enhance the beliefs or reduce the dependency on the beliefs already existing.

10) Monotonic reasoning -

- Advantages -
  1) In monotonic reasoning new axioms are asserted, new wff's may become provable, but no old proofs ever become invalid.
  2) It gives valid deduction which remains so always.
- Drawbacks -
  1) It is not possible to describe many envisioned or real world concepts; that is, it is limited in expressive power.
  2) There is no way to express uncertain, imprecise, hypothetical or vague knowledge, only the truth or falsity of such statements.

3) Available inference methods are known to be inefficient.

4) There is no way to produce new knowledge about world. It is only possible to add what is derivable from the axioms and theorems in the knowledge base.

6.5.2 Nonmonotonic Reasoning

1) It is the reasoning in which the axioms and/or the rules of inference are extended to make it possible to reason with incomplete information.

2) These system preserve, however, the property that, at any given moment, a statement is either believed to be true, believed to be false, or not believed to be either.

3) AI systems provide solutions for those problems whose facts and rules of inference are explicitly stored in the database and knowledge base. But data and knowledge are incomplete in nature and generally default assumptions are made.

4) For example, if we say that "Somu is a bird" the conclusion the arrive (by default) is that Somu can fly. But it is not necessary that Somu can fly. May be it cannot fly because of many reasons such as -

   i) Somu could be an Ostrich.

   ii) Somu's wings are broken.

   iii) Somu is too weak to fly.

   iv) Somu could be caged.

   v) Somu could be dead bird etc.

5) So, if one says that "Somu is a bird" by default conclusion is that Somu can fly. But if one adds a statement that "Somu is a Ostrich" then this added statement retract the previous made assumption. This is the basic logic behind nonmonotonic reasoning.

6) It means that new facts become known which contradict and invalidate old knowledge. The old knowledge was retracted causing other dependent knowledge to become invalid, thereby requiring further retractions. The retractions led to a shrinkage or nonmonotonic growth in the knowledge at times.

7) Non-monotonic reasoning -

- Advantages -

   1) In this logic, axioms and/or rules of inference are extended to make it possible to reason with incomplete information.

2) It can express uncertain, imprecise, hypothetical or vague knowledge. It preserves, however the property that at any instant, a statement is either true or false or not believed to be either.

- Drawbacks -
  1) Adding a new assertion may invalidate the old inference that depended on the absence of that assertion.
  2) The old knowledge is retracted causing other dependent knowledge to become invalid.
- Comparison (Monotonic and Non-Monotonic Reasoning) -

The conclusions derived using monotonic logics are valid deductions and they remain so. Adding new axioms increases the amount of knowledge contained in the knowledge base. Therefore, the set of facts and inferences in such systems can only grow larger, they cannot be reduced, that is they increase monotonically.

Whereas, in non-monotonic logic, new facts became known which contradicted and invalidated old knowledge. The old knowledge was retracted causing other dependent knowledge to become invalid, thereby requiring further retractions. The retractions led to a shrinkage or non-monotonic growth in the knowledge.

Solved Examples

Example 6.1 *Convert the following sentences to predicate logic -*

i) Marcus was a man ii) Marcus was a pompeian iii) All pompeians were Romans iv) Caesar was a ruler v) All romans were either loyal to caesar or hated him vi) Everyone is loyal to someone vii) People only try to assassinate rulers they are not loyal to viii) Marcus tried to assassinate caesar.

Solution : Set of wff's in predicate logic are as follows -

i) Marcus was a Man - Man (Marcus)

The representation captures the critical fact of Marcus being a man, but if fails to capture some of the information in the english sentence, namely the notion of past tense, i.e., one cannot conclude from above representation that a Marcus was a man, or Marcus is a man, or Marcus will be a man etc. Whether this omission is acceptable or not depends on the use to which we intend to put the knowledge. For this example, it will be all right.

ii) Marcus was a Pompeian

Pompeian (Marcus) - Its explanation is same as of first one.

iii) All pompeians were Romans

$\forall x$: Pompeian (x) $\rightarrow$ Roman (x)

In this presentation universal quantifier $\forall$ is used to present universal value of x. i.e., for all x in universe.

Pompeian (x) indicates that x is a pompeian. Next symbol '$\rightarrow$' is implies i.e., means and roman (x) indicates that x is Roman.

In other words we say that all x i.e., all pompeians were Roman.

iv) Caesar was a ruler

ruler (Caesar)

Here, one can ignore the fact that proper names are often not references to unique individuals, since many people share the same name. Sometimes deciding which of several people of the same name is being referred to in a particular statement may require a fair amount of knowledge and reasoning.

v) "All romans were either loyal to Caesar or hated him"

$\forall x$: Roman (x) $\rightarrow$ loyalto (x, Caesar) $\vee$ hate (x, Caesar)

In English, the word "OR" sometimes means the logical inclusive-OR and sometimes means the logical exclusive-OR (XOR).

Here, inclusive interpretation is used.

But, if one considers it as 'exclusive-OR' then presentation would be some what different as follows -

$\forall x$: Roman (x) $\rightarrow$ [loyalto (x, Caesar) $\vee$

hate (x, Caesar)) $\wedge$

$\neg$ (loyalto (x, Caesar) $\wedge$ hate (x, Caesar))]

vi) "Everyone is loyal to someone"

$\forall x : \exists y$: loyalto (x, y).

Here, one can use both the quantifiers i.e. $\forall \rightarrow$ universal quantifier and $\exists \rightarrow$ existential quantifier.

So, one can read above presentation as for all x, there exists a y such that x is loyal to y or indirectly we say

"Everyone is loyal to someone".

But here one can understand a different meaning i.e. there is someone to whom everyone is loyal, which would be written as

$\exists y : \forall x$: loyalto (x, y)

Now read it as, there exists a y, such that for all x, x is loyal to y.

Or

"everyone is loyal to y".

vii) "People only try to assassinate rulers they are not loyal to"

$\forall x : \forall y$: person (x) $\wedge$ ruler (y) $\wedge$

tryassassinate (x, y) $\rightarrow \neg$ loyal to (x, y)

This sentence, too, is ambiguous. Does it mean that the only rulers that people try to assassinate are those to whom they are not loyal, one above presented or does it mean that the only thing people try to do is to assassinate rulers to whom they are not loyal ?

viii) "Marcus tried to assassinate Caesar"

tryassassinate (Marcus, Caesar)

Example 6.2 *Convert the following sentences to predicate logic -*

i) Marcus was a man ii) Alive means not dead iii) If x divides y and y divides z then x divides z.

Solution : i) Refer to example 6.1.

ii) Alive means not dead

$\forall x : \forall t$: [alive (x, t) $\rightarrow \neg$ dead (x, t)]

$\wedge$ [$\neg$ dead (x, t) $\rightarrow$ alive (x, t)]

This is not strictly correct, since $\neg$ dead implies alive only for animate objects.

iii) If x divides y and y divides z then x divides z.

$\exists x : \exists y : \exists z$: [divides (x, y) $\wedge$ divides (y, z)] $\rightarrow$ divides (x, z)

Example 6.3 *Suppose you are given some facts about Marcus as follows -*

i) Marcus was a man. ii) Marcus was pompeian. iii) Marcus was born in 40 A.D. iv) All men are mortal. v) All pompeians died when the volcano erupted in 79 A.D. vi) No mortal lives longer than 150 years. vii) It is now 1991.

Now, answer the question "Is Marcus alive"? By proving.

Solution : Representing facts in predicate logic :

i) Marcus was a man.

man (Marcus)

Again we ignore the issue of tense.

ii) Marcus was a pompeian.

Pompeian (Marcus).

iii) Marcus was born in 40 A.D.

born (Marcus, 40).

iv) All men are mortal.

$\forall x$: man (x) $\rightarrow$ mortal (x)

v) All pompeians died when the volcano erupted in 79 A.D.

erupted (volcano, 79) $\wedge \forall x$: [Pompeian (x) $\rightarrow$ died (x, 79)]

vi) No mortal lives longer than 150 years.

$\forall x : \forall t_1 : \forall t_2$: mortal (x) $\wedge$ born (x, t_1) $\wedge$ gt($lt_2 - t_1$, 150) $\rightarrow$ dead (x, t_2)

There are various ways that the content of this sentence could be expressed. For example, one could introduce a function age and assert that its value is never greater than 150.

vii) It is now 1991.

now = 1991.

Here, exploit the idea of equal quantities that can be substituted for each other.

By using above facts we try to answer that whether "Marcus is alive or dead".

This can be done in two ways.

a) Prove that Marcus is dead because he was killed by the volcano.

b) Show that if Marcus is alive today, then it must of age more than 150 years, which is not possible.

To prove this additional knowledge is required.

For example.

viii) Alive means not dead.

$\forall x : \forall t$: [alive (x, t) $\rightarrow \neg$ dead (x, t)]

$\wedge$ [$\neg$ dead (x, t) $\rightarrow$ alive (x, t)]

ix) If someone dies, then he is dead at all later times.

$\forall x : \forall t_1 : \forall t_2$: died (x, t_1) $\wedge$ gt(t_2, t_1) $\rightarrow$ dead (x, t_2).

This representation says that one is dead in all years after the one in which one died. It ignores the question of whether one is dead in the year in which one died.

To answer that requires breaking time up into smaller units than years.

If this is done then one can add rules that say such things as "one is dead at the (year1, month1) if one died during (year1, month2) and month 2 precedes month 1". One can extend this to days and hours etc.

Fig. 6.1 and 6.2 shows these proofs by different ways.

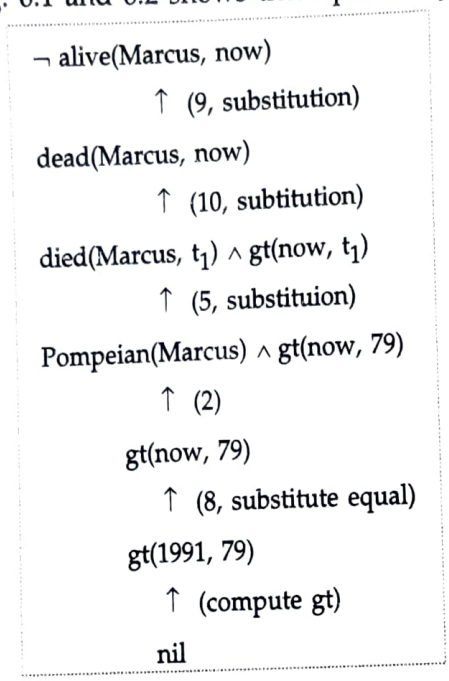


Fig. 6.1 One way of proving that Marcus is dead

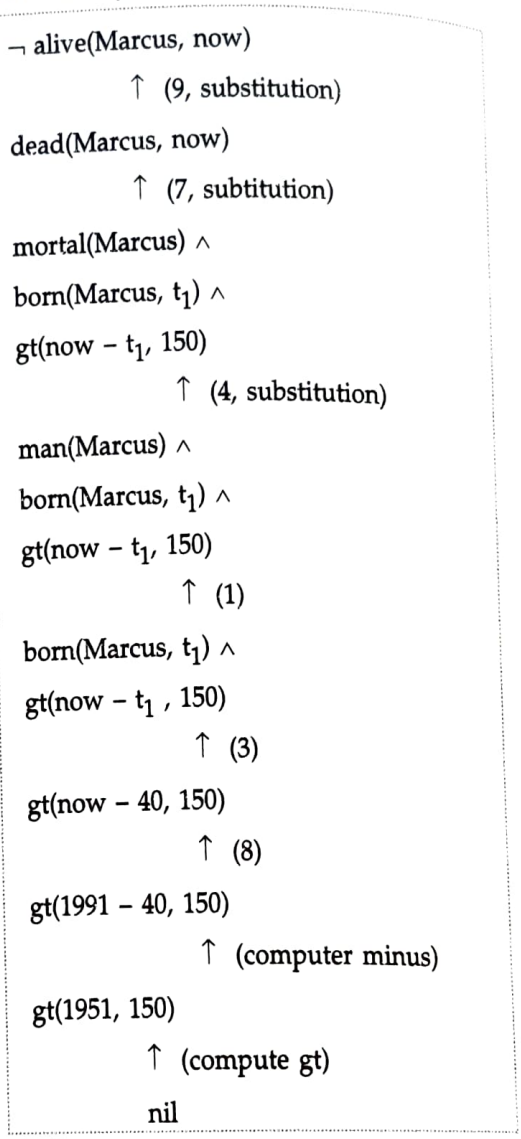


Fig. 6.2 Another way of proving that Marcus is dead

- From looking at the proofs just shown, two things should be clear -
  a) Every very simple conclusions can require many steps to prove.
  b) A variety of processes, such as matching, substitution and application of modus ponens are involved in the production of a proof. This is true even for the simple statements we are using.

Example 6.4 *Consider the following set of sentences -*

i) Marcus was a man ii) Marcus was a pompeian iii) All pompeians were Romans iv) Caesar was a ruler v) All romans were either loyalto caesar or hated him vi) Everyone is loyal to someone vii) People only try to assassinate rulers they are not loyal to viii) Marcus tried to assassinate caesar.

Prove that was Marcus loyal to caesar.

Solution : All the given sentences in predicate logic are given in example 6.1.

Now, by using the above facts one need to answer that, was marcus loyal to caesar or not.

Let's try to produce a formal proof, reasoning backward from the desired goal -

¬ loyal to (Marcus, Caesar)

In order to prove the goal, one need to use the rules of inference to transform it into another goal that can in turn be transformed and so on, until there are no satisfied goals remaining.

But there is a problem, that is, although we know that Marcus was a man, one do not have any way to conclude that Marcus was a person. So one add this fact.

ix) Marcus was a person

$\forall x$: man (x) $\rightarrow$ person (x)

Now one can satisfy the goal and produce proof that Marcus was not loyal to Caesar (as seen from rule (vii) and (viii))

Proof -

¬ loyalto (Marcus, Caesar)

↑ (7, substitution)

person (Marcus) ∧

ruler (Caesar) ∧

tryassassinate (Marcus, Caesar)

↑ (4)

person (Marcus)

tryassassinate (Marcus, Caesar)

↑ (8)

person (Marcus)

Example 6.5 *Consider the following sentences :*

i) If X is on top of y then y supports X

ii) If X is above y and they are touching each other then x is on top of y.

iii) A cup is above a book.

iv) A cup is touching a book

Transtate these statements in predicate logic. Prove by backward reasoning "Book Support Cup".

Solution : Conversion of above statements in predicate logic as follows -

i) $\exists x : \exists y$: on-top-of (x, y) $\rightarrow$ support (y, x) ... (i)

ii) $\exists x : \exists y$: [above (x, y) $\wedge$ touch (x, y) $\wedge$ touch (y, x)] $\rightarrow$ on-top-of (x, y) ... (ii)

iii) $\exists x : \exists y$: above (cup, book) ... (iii)

iv) $\exists x : \exists y$: touch (cup, book) ... (iv)

Proof by backward reasoning is as follows -

It predicate logic conversion will be -

Support (Book, cup)

$\uparrow$ (1, substitution)

Ontopof (Cup, Book)

$\uparrow$ (2, substitution)

above (Cup, Book) $\wedge$ touch (Cup, Book) $\wedge$ touch (Book, Cup)

(3, substitution)

touch (Cup, Book) $\wedge$ touch (Book, Cup)

$\uparrow$ (4 substitution)

Nill

The proof of given statement is as above. The term Nil at the end of proof indicate that the list of conditions remaining to be proved is empty and so the proof has succeeded.

Example 6.6 *Convert the following statement into clause form*

i) $(\forall x)\ (\exists y)\ (\forall z)\ [p(x, y) \wedge q(x, z) \rightarrow r(z, x)]$

ii) $(\forall x)\ (\forall y)\ [(\exists z)\ p(x, z) \wedge p(y, z)) \rightarrow (\exists u)\ Q(x, y, u)]$

Solution : i) Given statement is

$$(\forall x)\ (\exists y)\ (\forall z)\ [p(x, y) \wedge q(x, z) \rightarrow r(z, x)]$$

By applying rule 1,

$$(\forall x(\ (\exists y)\ (\vee z)\ [\sim(p(x, y) \wedge q(x, y)) \vee r(z, x)]$$

By applying rule 2,

$$(\forall x)\ (\exists y)\ (\forall z)\ [\sim p(x, y) \vee \sim q(x, z) \vee r(z, x)]$$

Rule 3 and 4 is not required here, so by applying rule 5,

$$(\forall x)\ (\forall z)\ [\sim p\ (x, A(x)) \vee \sim q(x, z) \vee r(z, x)]$$

By applying rule 6,

$$[\sim p(x, A(x)) \vee \sim q(x, z) \vee r(z, x)]$$

By applying rule 7,

$$\sim p(x, A(x)) \vee \sim q(x, z) \vee r(z, x)$$

The above expression is a clausal form.

ii) Given statement is

$$(\forall x)\ (\forall y)\ [((\exists z)\ p(x, z) \wedge p(y, z)) \rightarrow (\exists u)\ Q(x, y, u)]$$

By applying rule 1,

$$(\forall x)\ (\forall y)\ [\sim((\exists z)\ p(x, z) \wedge p(y, z)) \vee (\exists u)\ Q(x, y, u)]$$

By applying rule 2,

$$(\forall x)\ (\forall y)\ [((\forall z)\ \sim p(x, z) \vee \sim p(y, z)) \vee (\exists u)\ Q(x, y, u)]$$

By applying rule 4,

$$(\forall x)\ (\forall y)\ (\forall z)\ (\exists u)\ [((\sim p(x, z) \vee \sim p(y, z)) \vee (\exists u)\ Q(x, y, u)]$$

By applying rule 5,

$$(\forall x)\ (\forall y)\ [((\sim p(x, A(x, y)) \vee \sim p(y, A(x, y))) \vee Q(x, A(x, y), B(x, y))]$$

By applying rule 6,

$$[((\sim p(x, A(x, y)) \vee \sim p(y, A(x, y))) \vee Q(x, A(x, y), B(x, y))]$$

By applying rule 7,

$$\sim p(x, A(x, y)) \vee \sim p(y, A(x, y) \vee Q(x, A(x, y), B(x, y))$$

The above expression is a clausal form.

Example 6.7 *Explain the following terms in common sense -*

i) Ribbon (Object, Side$_1$, Side$_2$) ii) Along (x, y) iii) Across (x, y).

Solution : i) Ribbon (Object, Side$_1$, Side$_2$) -

A ribbon is essentially a curve viewed at a coarser level of granularity, resulting in a two-dimensional ribbon like shape.

The problem world contains many objects that are usefully viewed as ribbons, e.g., rivers and bridges (Fig. 6.3).

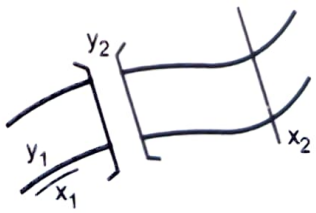


Fig. 6.3 Two ribbons (y_1 and y_2) and Two curves (x_1 and x_2)

$$\text{TERMINAL}(P, c) \equiv$$
$$\text{INSIDE}(P, c) \wedge$$
$$\forall c_1, c_2 : \text{INSIDE}(c_1, c) \wedge \text{INSIDE}(c_2, c)$$
$$\wedge \text{INSIDE}(P, c_1) \wedge \text{INSIDE}(p, c_2)$$
$$\rightarrow \text{INSIDE}(c_1, c_2) \vee \text{INSIDE}(c_2, c_1)$$

It is the definition of a terminal point P of a curve c.

ii) Along (x, y) -

$$\text{ALONG}(x, y) \equiv \text{CURVE}(x) \wedge \text{RIBBON}(y, s_1, s_2) \wedge$$
$$\forall_z : \text{INSIDE}(z, x) \rightarrow \text{ADJACENT}(z, y)$$

iii) Across (x, y) -

$$\text{ACROSS}(x, y) \equiv \text{CURVE}(x) \wedge \text{RIBBON}(y, s_1, s_2) \wedge$$
$$\text{PERPENDICULAR}(x, \text{AXIS}(y, s_1, s_2)) \wedge$$
$$\text{ADJACENT}(x, y) \wedge \text{ADJACENT}(x, s_1) \wedge$$
$$\text{ADJACENT}(x, s_2)$$

These definitions assume that the terms PERPENDICULAR, AXIS and ADJACENT have been defined and that the commonsense axiom that an object x is ADJACENT to an object y if any part of x is ADJACENT to y, are supplied.

A robot could use the ALONG relation to plot a course down the river's edge. It could similarly use the ACROSS relation to navigate to the other side of the river. Unfortunately, the ACROSS relation is not enough, as the robot might try to cross the river without using the bridge.

Example 6.8 *Consider the following sentences -*

a) Krishna like all kind of food. b) Apples are food. c) Bread is food. d) Anything anyone eat and is not killed by is food. e) Ravi eats peanuts and is still alive. f) Chetan eat everything Ravi eats. g) Translate these sentences into formula in predicate logic. h) Prove that Krishna likes peanuts using resolution.

Solution : i) Translation in Predicate logic -

a) Krishna like all kind of food.

$\forall x$: food (x) $\rightarrow$ like (Krishna, x)

b) Apples are food.

food (Apple)

c) Bread is food.

food (Bread)

d) Anything everyone eat and is not killed by is food

$\forall x : \forall y$: eat (x, y) $\wedge$ ~ kill (x, y) $\rightarrow$ food (y)

e) Ravi eats peanuts and is still alive

eat (Ravi, peanut) $\wedge$ alive (Ravi)

f) Chetan eat everything Ravi eats

$\forall x$: eat (Chetan, x) $\rightarrow$ eat (Ravi, x)

ii) Proof by resolution - Need to prove that like (Krishna, peanut)

- Assume the negation of the result

  ~ like (Krishna, peanut) ... (i)

- The given axioms are

  $\forall x$: food(x) $\rightarrow$ like(Krishna, x) ... (ii)

  food(Apple) ... (iii)

  food(Bread) ... (iv)

  $\forall x : \forall y$: eat(x, y) $\wedge$ ~ kill(x, y) $\rightarrow$ food(y) ... (v)

  eat(Ravi, peanut) $\wedge$ alive (Ravi) ... (vi)

  $\forall x$: eat(Chetan, x) $\rightarrow$ eat(Ravi, x) ... (vii)

- Equation (ii) can be written as

  ~ food(x) $\vee$ like(Krishna, x) ... (viii)

- In equation (viii), substitute x = peanut

  ~ food(peanut) $\vee$ like(Krishna, peanut) ... (ix)

- Resolving equations (i) and (ix), we have

  ~ food(peanut) ... (x)

- Resolving equations (iii) and (x) contradiction arrives. Hence, the negation of the result is false or the result is true.

Example 6.9 *Convert the following sentences to predicate logic, clause form and thus prove by resolution that - "Marcus was a Roman"*

i) Marcus was a Pompeian. ii) All Pompeian were Romans.

Solution : In Predicate logic -

i) Marcus was a Pompeian.

Pompeian (Marcus)

ii) All pompeians were Romans.

$\forall x$: Pompeian (x) $\rightarrow$ Roman (x)

In clause form -

i) Pompeian (Marcus).

ii) ¬ Pompeian (x_1) ∨ Roman (x_1)

To prove -

Roman (marcus)

Proof -

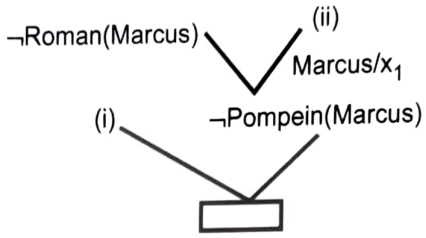


Fig. 6.4

Empty clause means that the clause which initially taken is totally contradict and cannot be true, which results that

¬ Roman (marcus) is FALSE or

Roman (marcus) is TRUE. **Hence proved.**

Example 6.10 *Convert the following sentences into propositional form :*

Given :

Sentence	*Symbol*
If rains	*r*
Picnic is cancelled	*pc*
be wet	*wet*
stay at home	*s*

i) I will be wet if it rains

ii) If it rains but I stay at home I won't be wet

iii) If it rains and the picnic is not cancelled or I don't stay at home, I will be wet.

iv) Either it does not rain or I am staying at home

v) Whether or not the picnic is cancelled, I am staying at home, if it rains

Solution : 1. Propositional form :

i) $r \rightarrow wet$

ii) $(r \vee s) \rightarrow wet$

iii) $((r \wedge \neg pc) \vee \neg s) \rightarrow wet$

iv) $\neg r \vee s$

v) $(pc \vee \neg pc) \wedge r \rightarrow s$

Example 6.11 *Convert the following sentences into FOL representation*

a) Some students took English in spring 2004

b) Every student who takes English passes it

c) John is a man

d) Father of Bill is a teacher

e) John likes books and music

f) John lives in a yellow house

g) If the car belongs to John then it is green

h) All elephants are gray in color

i) John likes all kind of foods

j) Anything any one eats and isn't killed by is food.

k) Sue eats everything Bill eats

l) Not all student take both History and Biology

m) Only one student failed in History

Solution :

a) $\exists x$ Student (x) $\wedge$ Take (x, English, Spring2004)

b) $\forall x$ Student (x) $\wedge$ Take (x, English)=>Passes (x, English)

c) Man (John)

d) Teacher (Father of (Bill))

e) Likes (John, Books) $\wedge$ Likes (John, Music)

f) Lives (John, House) $\wedge$ Color (House, Yellow)

(or)

Lives ((John, Color(House, Yellow))

g) Belongs (Car, John)=> Color(Car, Green)

h) $\forall x$ Elephant (x)=> Color (x, Gray)

i) $\forall x$ Likes (John, x) $\wedge$ Food(x)

j) $\forall x \forall y$ Person(x) $\wedge$ Food(y) $\wedge$ Eats(x, y) $\wedge \neg$ Killed(x)=>Isfood(x)

k) $\forall x$ Eats (Sue, x) = Eats (Bill, x)

l) $\exists x$ student (x) $\wedge$ Take(x, History) $\wedge$ Take(x, Biology)

m) $\exists x$ student (x) $\wedge$ Failed (x, History) $\wedge \forall y$ y $\neq$ x = $\neg$ Failed (y, History)

Example 6.12 *I. From the following KB, solve the given task using any one of the resolution techniques.*

1. Mani likes easy games 2. Boxing is hard 3. All the indoor games are easy

4. Table Tennis is an indoor game.

Task : Find the name of the game. which is liked by Mani.

II. From the following KB, solve the given task using any one of the resolution techniques for INF and CNF representation.

1. Jack owns a dog 2. Every dog owner is an animal lover 3. No animal lover kills an animal 4. Either Jack or Curiosity killed the cat, who is named Tuna 5. Tuna is a cat 6. All cats are animals

Task : Show that Kills (Curiosity, Tuna) is true.

III. For each pair of atomic sentences, give the most general unifiers if it exists.

a) P(A, B, B), P(x, y, z) b) Q(y, G(A, B)), Q(G(x, y), y)

Solution : I. FOL representation :

1. $\forall x$ Easygame (x) $\rightarrow$ Likes (Mani, Game (x))
2. Hard (Boxing)
3. $\forall x$ Indoorgame (x) $\rightarrow$ Easygame (x)
4. Indoorgame (TableTennis)

CNF representation :

1. ~Easygame (x) $\vee$ Likes (Mani, Game (x))
2. Hard (Boxing)
3. ~Indoorgame (x) $\vee$ Easygame (x)
4. Indoorgame (TableTennis)

Proof :

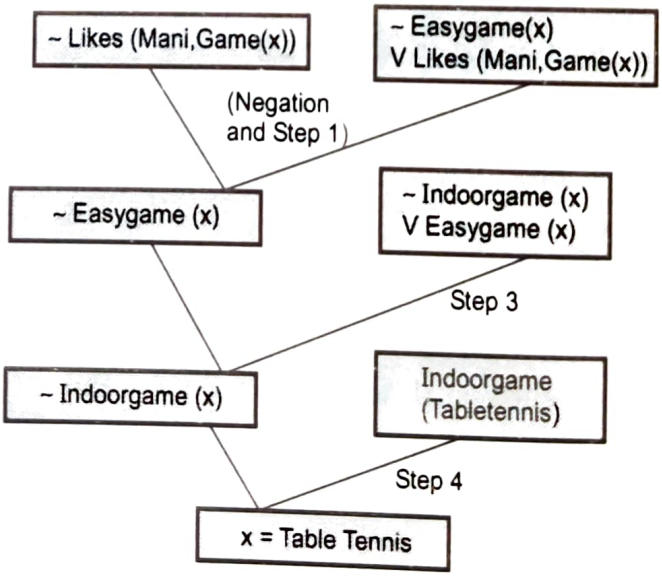


Fig. 6.5 (a)

$\therefore$ Mani like TableTennis

II. FOL representation :

1. $\exists x$ Dog (x) $\wedge$ Owns (Jack, x)
2. $\forall x$ ($\exists y$ Dog (y) $\wedge$ Owns (x, y)) $\Rightarrow$ Animallover (x)
3. $\forall x$ Animallover (x) $\Rightarrow$ $\forall y$ Animal (y) $\Rightarrow \neg$ Kills (x, y)
4. Kills (Jack, Tuna) $\vee$ Kills (Curiosity, Tuna)
5. Cat (Tuna)
6. $\forall x$ Cat (x) $\Rightarrow$ Animal (x)

INF representation :

1. a. Dog (D)

   b. Owns (Jack, D)
2. Dog (y) $\wedge$ ~Owns (x, y) $\Rightarrow$ Animallover (x)
3. Animallover (x) $\wedge$ Animal (y) $\wedge$ Kills (x, y) $\Rightarrow$ False
4. Kills (Jack, Tuna) $\vee$ Kills (Curiosity, Tuna)
5. Cat (Tuna)
6. Cat (x) $\Rightarrow$ Animal (x)

CNF representation :

1. a. Dog (D)

   b. Owns (Jack, D)
2. ~ Dog (y) $\vee$ ~ Owns (x, y) $\vee$ Animallover (x)
3. ~ Animallover (x) $\vee$ ~ Animal (y) $\vee$ ~ Kills (x, y)
4. Kills (Jack, Tuna) $\vee$ Kills (Curiosity, Tuna)
5. Cat (Tuna)
6. ~ Cat (x) $\vee$ Animal (x)

Resolution with refutation using INF representation :

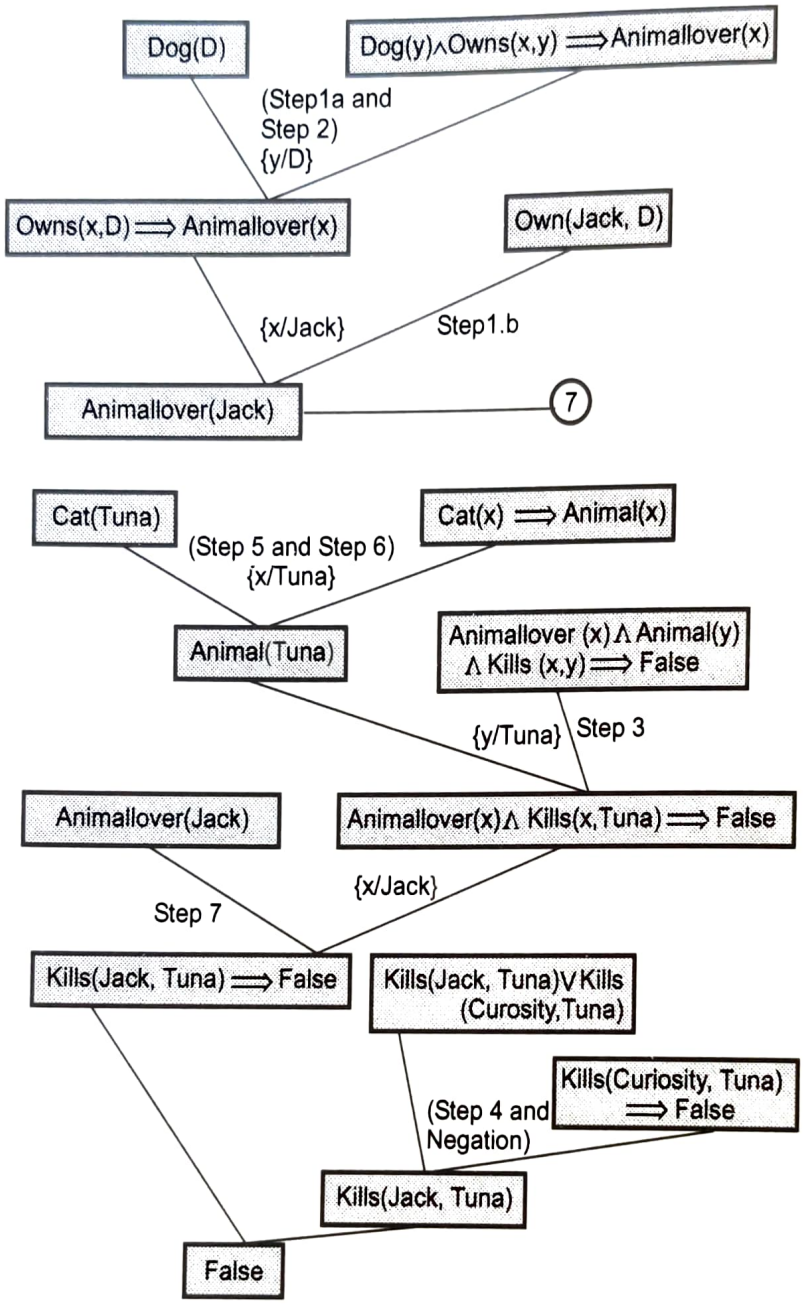


Fig. 6.5 (b)

To solve the given task we assumed the negation (Kills (Curiosity, Tuna) ⇒ False) and using inference rules we derive a contradiction, False, which means that the assumption must be false, and Kills (Curiosity, Tuna) is true.

Resolution with refutation using CNF representation :

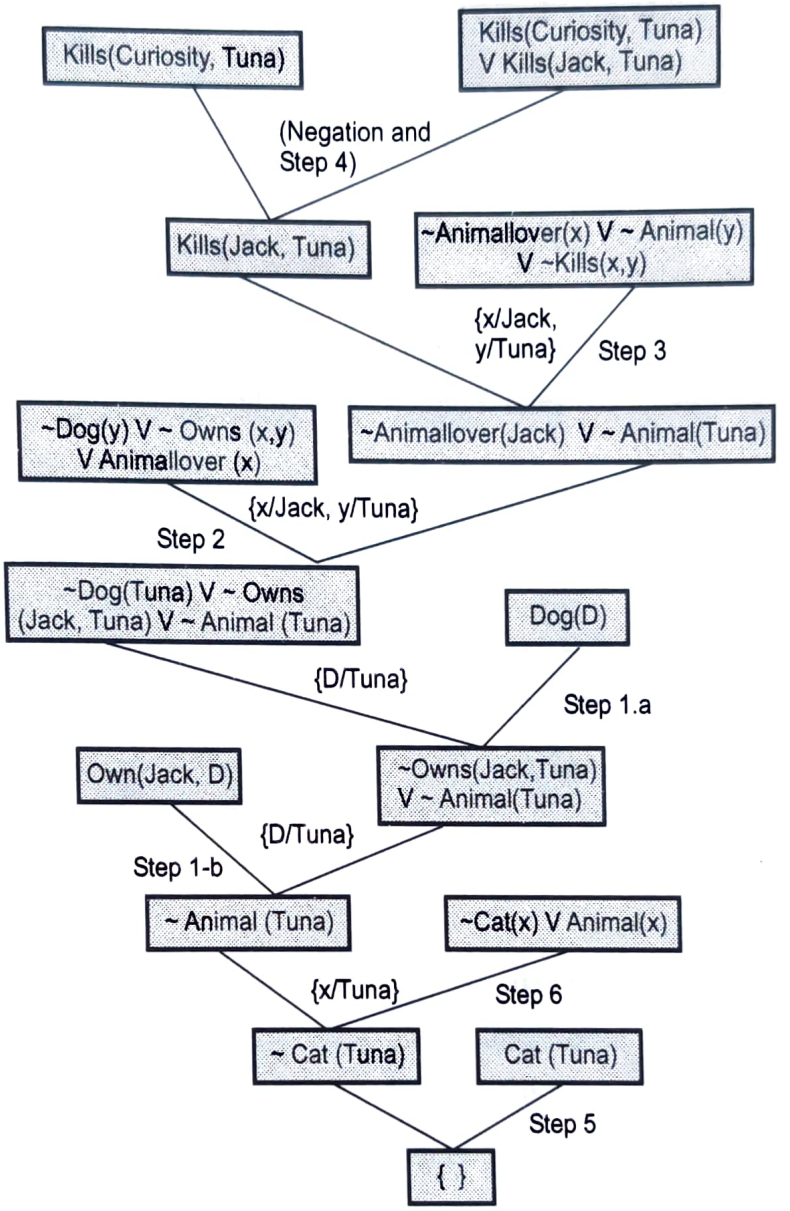


Fig. 6.5 (c)

To prove the given task, we assumed the negation of it (i.e.) ~ Kills (Curiosity, Tuna) and using inference rule we derive a contradiction as empty set, which means that the assumption must be false, and Kills (Curiosity, Tuna) is true.

III. a. {x | A, y | B, z | B}

b. No unifer (x cannot bind to both A and B)

Example 6.13 *Represent the following sentences in FOL :*

a) Perrier is a kind of water

b) John has perrier in his water bottle

c) Water is a liquid between 0 and 100 degrees

d) The water in John's water bottle is frozen.

Solution : a) Perrier $\subset$ Water

b) $\exists b \; \forall \; w \; w \in \text{Water} \wedge b \in \text{WaterBottles} \wedge \text{Has (John, b, Now)} \wedge \text{Inside (w, b, Now)} \Rightarrow w \in \text{Perrier.}$

c) $\forall \; w \; w \in \text{Water} \Rightarrow (\text{Degree (0)} < \text{Temperature (w)} < \text{Degree (100)}) \Leftrightarrow w \in \text{Liquid.}$

d) $\exists b \; \forall \; w \; w \in \text{Water} \wedge b \in \text{WaterBottles} \wedge \text{Has (John, b, Now)} \wedge \text{Inside (w, b, Now)} \Rightarrow (w \in \text{Solid, Now})$

Example 6.14 *Consider the following facts :*

- *Steve only likes easy courses.*
- *Science courses are hard.*
- *All the courses in the HaveFun department are easy.*
- *BK301 is a HaveFun department course.*

Use resolution to answer the question "What course would Steve like"?

Solution : The following statements are assumed to be true :

1. Steve only likes easy courses.
2. Science courses are hard.
3. All the courses in the HaveFun department are easy.
4. BK301 is a HaveFun deparment course.

To answer : What course would Steve like ?

The predicate logic encoding of the premises of the previous problem is as follows :

1. forall x easy(x) -> likes(steve,x)
2. forall x science(x) -> ~easy(x)
3. forall x HaveFun(x) -> easy(x)
4. HaveFun(BK301)

The conclusion is encoded as

likes(steve,x).

First put premises in the clause form and the negation of conclusion to the set of clauses.

- (1) ~easy(x) or likes(steve,x)
- (2) ~science(x) or ~easy(x)

- (3) ~science(x) or ~easy(x)
- (4) ~HaveFun(x) or easy(x)
- (5) HaveFun(BK301)
- (6) ~likes(steve,x)

A resolution proof may be obtained by the following sequence of resolutions (each step includes a parenthesized number of the resolvent generated in the current step; 1 and 5 means that we resolve clauses (1) and (5)):

- (7) 1 and 6 yields resolvent ~easy(x).
- (8) 4 and 7 yields resolvent ~basketweaving(x).
- (9) 5 and 8 yields empty clause; the substitution x/BK301 is produced by the unification algorithm which says that the only wff of the form likes(steve,x) which follows from the premises is

  likes(steve,BK301).

Thus, resolution gives us a way to find additional assumptions (in this case x=BK301) which make theorem true.

Answer in Brief

1. *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.* **(Refer section 6.1.8)**
2. *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.* **(Refer section 6.18)**
3. *Give resolution in propositional logic.* **(Refer section 6.4)**
4. *Write down and explain the unification algorithm in predicate logic.*
   (Refer section 6.4)

   OR

   Explain unification algorithm.

   OR

   Write short note on unification algorithm.
5. *Differentiate the monotonic and non-monotonic reasoning.* **(Refer section 6.5)**
6. *Name two standard quantifier.*

Ans. : The two standard quantifier are universal quantifiers (represented as ∀, which means "For all") and existential quantifier (represented as ∃, which means "there exists some object").

They are used for expressing properties of entire collection of objects rather than just a single object.

For example :

i) ∀ x Happy (x) means that "if the universe of discourse is people, then everyone is happy".

ii) ∃ x Happy (x) means that "if the universe of discourse is people, then this means that there is at least one happy person".

7. *What is a purpose of unification ?*

Ans. : It is used for finding substitutions for inference rules, which can make different logical expression to look identical. It helps to match to logical expressions. Therefore it is used in many algorithm in first order logic.

8. *What is ontological commitment (what exists in the world) of first order logic ? Represent the sentence "Brothers are siblings" in first order logic.*

Ans. : Ontological commitment means what assumptions language makes about the nature if reality.

Representation of "Brothers are siblings" in first order logic is

∀ x, y [Brother (x, y) ⇒ Siblings (x, y)].

9. *Diffentiate between propositional versus first order predicate logic.*

OR Distinguish between predicate logic and propositional logic.

OR Diffentiate propositional and first order logic.

Ans. : Following are the comparative differences between propositional logic and first order logic

1) Propositional logic is less expressive and do not reflect individual object's properties explicitely.

   First order logic is more expressive and can represent individual object along with all its properties.

2) Propositional logic can not represent relationship among objects whereas first order logic can represent relationship.

3) Propositional logic do not consider generalization of objects where as first order logic handles generalization.

4) Propositional logic includes sentence letters (A, B, C) and logical connectives, but not quantifier.

   First order logic has the same connectives as propositional logic, but it also has variables for individual objects, quantifier, symbols for functions, and symbols for relations.

10. *What factors justify whether the reasoning is to be done in forward or backward reasoning ?*

Ans. : Following factors justify whether the reasoning is to be done in forward or backward reasoning -

1) Which state is more possible to begin with, the start state or goal state ?

2) Is there a need to justify the reasoning ?

3) What kind of events trigger the problem - solving ?

4) In which direction is the branching factor greatest ? One should go in the direction with lower branching factor.

11. Define diagnostic rules with example.

Ans. : Diagnostic rules are used in first order logic for inferencing. The diagnostic rules generate hidden causes from observed effect. They help to deduce hidden facts in the world. For example consider the wumpus world.

The diagnostic rule finding 'pit' is,

"If square is breezy some adjacent square must contain pit", which is written as,

$\forall s$ Breezy(s) $\Rightarrow \exists_r$ Adjacent (r,s) $\wedge$ pit(r)

12. Represent the following sentence in predicate form "All the children likes sweets".

Ans. : $\forall x$ child(x) $\wedge$ sweet(y) $\wedge$ likes (x,y).

13. What is skolemization ?

Ans. : Skolemization is the process of removing existential quantifier by elimination. It converts a sentence with existential quantifier into a sentence without existential quantifier such that the first sentence is satisfiable if and only if the second is.

For eliminating an existential quantifier, each occurance of its variable is replaced by a skolem function whose argument are the variables of universal quantifier whose scope includes the scope of the existential quantifier.

14. Define the first order definite clause.

Ans. : 1) They are disjunctions of literals of which exactly one is positive.

2) A definite clause is either atomic sentence or is an implication whose antecedants (left hand side clause) is a conjunction of positive literals and consequent (Right hand side clause) is a single positive literal.

For example :

Princess (x) $\wedge$ Beautiful (x) $\Rightarrow$ GoodHearted (x)

Princess (x)

Beautiful (x)

15. Write the generalized modus ponens rule.

OR State generalized modus ponens.

Ans. : 1) Modus ponens :

If the sentence **P** and **P -> Q** are known to be true, then modus ponens lets us infer **Q**.

For example : If we have statment, " If it is raining then the ground will be wet" and "It is raining". If P denotes " It is raining" and Q is "The ground is wet" then the first expression becomes $P \rightarrow Q$. Because it is indeed now raining (P is true), our set of axioms becomes,

$\{ P \rightarrow Q$

$P \}$

Through an application of modus ponens, the fact that "The ground is wet" (Q) may be added to the set of true expressions.

- **The generalized modus ponens :**

  For atomic sentences P_i, P_i', and q, where there is a substitution Q such that SUBST $(\theta, P_i') =$ SUBST (θ, P_i'),

  For all i,

$$\frac{P_1', P_2', \ldots P_n', (P_1 \wedge P_2 \wedge \ldots \wedge P_n \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

  There are n+1 premises to this rule : - The 'n' atomic sentences p_i' and the one implication. The conclusion is the result applying the substitution θ to the consequent q.

Q.16 Define atomic sentence and complex sentence.

Ans. :

i. An atomic sentence is formed from a predicate symbol followed by a parenthesized list of terms.

   For example : Stepsister (Cindrella, Drizella)

ii. Atomic sentences can have complex terms as the arguments.

   For example : Married (Father (Cindrella), Mother(Drizella))

iii. Atomic sentences are also called atomic expressions, atoms or propositions.
   For example : Equal (plus (two, three), five) is an atomic sentence.

Complex sentences

i. Atomic sentences can be connected to each other to form complex sentence.

   Logical connectives, $\wedge, \vee, \neg, \rightarrow$ can be used to connect atomic sentences.

   For example :

   $\neg$ Princess (Drizella) $\rightarrow$ Princess (Cindrella)

ii. (foo (two, two, plus (two, three))) $\rightarrow$ (equal (plus (three, two), five) $\equiv$ true) is a sentence because all its components are sentences, appropriatly connected by logical operators.

- **Various sentences in first order logic formed using connectives :**

1) If S is a sentence, then so is its negation, $\neg S$.
2) If S_1, and S_2 are sentences, then so is their conjunction, $S_1 \wedge S_2$.
3) If S_1 and S_2 are sentences, then so is their disjunction, $S_1 \vee S_2$.
4) If S_1 and S_2 are sentences, then so is their implication, $S_1 \rightarrow S_2$.
5) If S_1 and S_2 are sentences, then so is their equivalence, $S_1 \equiv S_2$.

17. What is unification ?

Ans. : 1) It is the process of finding substitutions for lifted inference rules, which can make different logical expression to look similar (identical).

2) Unification is an procedure for determining substitutions needed to make two first order logic expressions match.

3) Unification is important component of all first order logic inference algorithms.

4) The unification algorithm takes two sentences and returns a unifier for them, if one exists.

18. Differentiate forward chaining and backward chaining.

Ans. : 1) Forward chaining is data driven.

- It is automatic unconscious processing.
- Example - Object recognition, routine decisions.
- It may do lots of work that is irrelevant to the goal.

2) Backward chaining is goal driven.

- It is appropriate for problem solving.
- Example : Where are my keys ?,

  How do I get into a PhD programme ?
- Complexity of backward chaining can be much less than linear in size of knowledgebase.

19. Define meta rules.

Ans. : The rules that determine the conflict resolution strategy are called meta rules. Meta rules define knowledge about how the system will work. For example, meta rules may define that knowledge from Expert 1 is to be trusted more than knowledge from Expert 2. Meta rules are treated by the system like normal rules, but are they are given higher priority.

20 Convert the following into Horn clauses.

$\forall x : \forall y : cat(x) \vee fish(y) \rightarrow likes\_to\_eat(x,y)$

Ans. : Horn clauses are as follows,

$$\neg \text{cat}(x) \vee \neg \textit{fish}(y) \vee \textit{likes\_to\_eat}\ (x,y)$$

21. *Discuss forward and backward chaining ?* **(Refer section 6.4)**
22. *Explain resolution procedure ?* **(Refer section 6.4)**
23. *Discuss the syntax and semantics of first order logic.* **(Refer section 6.1)**
24. *Describe the general process of knowledge engineering ?* **(Refer section 6.3)**
25. *Discuss forward and backward chaining with example ?* **(Refer section 6.4)**
26. *Describe forward chaining and backward chaining algorithm ?* **(Refer section 6.4)**
27. *Apply both algorithm to prove that "West is criminal"* **(Refer section 6.4)**
28. *What is conjunctive normal form of a rule ? Define Skolemization.* **(Refer section 6.4)**
29. *Describe in detail the steps involved in the knowledge engineering process ?* **(Refer section 6.3)**
30. *Explain Unification algorithm used for reasoning under predicate logic (first order logic) with an example.* **(Refer section 6.4)**
31. *Explain the forward chaining process in detail with example. What is the need of incremental chaining ?* **(Refer section 6.4)**
32. *Explain the inferencing process in first order logic, using suitable example.* **(Refer section 6.4)**
33. *Explain the steps involved in knowledge engineering process with example.* **(Refer section 6.3)**
34. *Discuss backward chaining algorithm.* **(Refer section 6.4)**
35. *Explain the algorithm for computing most general unifiers.* **(Refer section 6.4)**
36. *Explain with an example the use of unification algorithm to prove the concept of resolution.* **(Refer section 6.4)**
37. *Explain the forward chaining process and efficient forward chaining with example. State its usage.* **(Refer section 6.4.9)**
38. *State and explain the various steps in knowledge engineering process.* **(Refer section 6.3)**
39. *What are the steps to convert first order logic sentence to normal form ? Explain each step.* **(Refer section 6.4.11)**
40. *Represent the following sentences in predicate logic and convert the following sentences to CNF form.*
    (1) All women who like ice-creams like chocolates.
    (2) No man is happy with a spendthrift wife.
    (3) The best movie in Hollywood is always better than the best movie in Bollywood.
    (4) Some people like eating outside all the time and some people like eating at home all the time.

(5) *It might be argued that one aspect of intelligent behavior is the ability to infer new facts about the world by combining existing ones. Has the theory of logic given us a tool to allow computers to display this sort of intelligence ? Can humans make other leaps of inference that are impossible with logic alone ?* **(Refer section 6.4.11)**

41. *Differentiate propositional logic with FOL. List the inference rules along with suitable examples for first order logic.* **(Refer section 6.1)**

42. *Consider the following sentences :*
    (1) *John likes all kinds of food.*
    (2) *Apples are food.*
    (3) *Chicken is food.*
    (4) *Anything anyone eats and isn't killed alive.*
    (5) *Sue eats everything bill eats.*
    (A) *Translate these sentences into formulas in predicate logic.*
    (B) *Convert the formulas of a part into clause form.*
    (C) *Prove that "John likes peanuts" using forward chaining.*
    (D) *Prove that "John likes peanuts" using backward chaining.* **(Refer section 6.4.11)**

43. *Explain standard quantifiers of first order logic with example.* **(Refer section 6.4.3)**

44. *Explain the forward chaining algorithm with the help of the pseudo-code.* **(Refer section 6.4.9)**

45. *Give the completeness proof of resolution.* **(Refer section 6.4.11)**

46. *Explain forward chaining and backward chaining algorithm with an example.* **(Refer section 6.4.9)**

47. *Illustrate the use of first order logic to represent knowledge.* **(Refer section 6.1)**

48. *Write short note on unification.* **(Refer section 6.4.6)**

49. *Explain forward chaining and backward chaining algorithm with an example.* **(Refer section 6.4.9)**

50. *Illustrate the use of first order logic to represent knowledge.* **(Refer section 6.1)**

51. *Write short note on unification.* **(Refer section 6.4.6)**

52. *Explain in detail about forward and backward chaining with an example.* **(Refer section 6.4.9)**

53. *Consider the following sentences :*
    John likes all kinds of food
    Apples are food
    Chicken is food
    Anything anyone eats and isn't killed by is food
    Bill eats peanuts and is still alive
    Sue eats everything Bill eats.
    i) Translate these sentences into formulas in predicate logic.
    ii) Convert the formulas of part a into clause form. **(Refer section 6.4)**

54. *Trace the operation of the unifacination algorithm on each of the following pairs of literals :*
    i) f(Marcus) and f(Caesar)
    ii) f(x) and f(g(y))
    iii) f(Marcus, g(x,y)) and f(x, g(Caesar, Marcus) **(Refer section 6.4)**

55. *Consider the following facts :*
    - *Steve only likes easy courses.*
    - *Science courses are hard.*
    - *All the courses in the HaveFun department are easy.*
    - *BK301 is a HaveFun department course.*

    Use resolution to answer the question "What course would Steve like"?
    (Refer example 6.14)

56. *Relate first order logic with proposition logic and discuss in detail about the same.*
    (Refer section 6.1)

57. *Describe a procedure for converting a sentence to CNF with an example.*
    (Refer section 6.4)

58. *Explain forward chaining and backward chaining for proportional definite clauses.*
    (Refer section 6.4)

59. *Consider the following facts*
    1) All students in 4^{th} year are intelligent.
    2) Raja is a 4^{th} year student.
    3) Ragu is a 3^{rd} year student.
    4) 3^{rd} year studensts are not intelligent.
    5) 4^{th} year students have no friends in 3^{rd} year.
    Represent the facts in predicate convert to clause form and prove by resolution, "Raja is not friend of Ragu". **(Refer section 6.4)**

60. *Explain the unification algorithm with an example.* **(Refer section 6.4)**

61. *Consider the following facts*
    1) There are 5000 employees in XYZ company.
    2) Employees earning more than ₹ 25000/- annum pay tax.
    3) John is a manager in XYZ company.
    4) Manager earns ₹ 50,000/-.
    Represent the facts in predicate convert to clause form and prove by resolution, "John pays tax".
    (Refer section 6.4)

62. *Explain dempster shafer theory with an example.* **(Refer section 6.5)**
63. *What are fuzzy membership functions ? Explain them with examples.*
    (Refer section 6.5)

6.6 University Questions and Answers

Summer - 18

Q.1 *Discuss non-monotonic reasoning.* **(Refer section 6.5)** [3]

Q.2 *Explain following terms in reference to predicate logic Resolution.*
a. Unsuccessful attempt at resolution b. Equality c. Reduce
d. Trying several substitute **(Refer section 6.4)** [7]

Winter - 18

Q.3 *Explain the procedure to convert well formed formula to clause form with the help of example.* **(Refer section 6.1)** [7]

Q.4 *Differentiate monotonic and non-monotonic reasoning.* **(Refer section 6.5)** [4]

Q.5 *Explain resolution in predicate logic.* **(Refer section 6.4)** [7]

Summer - 19

Q.6 *Discuss non-monotonic reasoning.* **(Refer section 6.5)** [3]

Winter - 19

Q.7 *Write a note on non-monotonic reasoning.* **(Refer section 6.5)** [4]

Summer - 20

Q.8 *Define the following words in the context of AI : Logical reasoning.* **(Refer section 6.1)** [1]

Q.9 *Define predicate logic.* **(Refer section 6.1)** [2]

Q.10 *List out the property of monotonic and non monotonic reasoning.* **(Refer section 6.5)** [4]

❑❑❑