UNIT III KNOWLEDGE REPRESENTATION

First Order Predicate Logic – Prolog Programming – Unification – Forward Chaining-Backward Chaining – Resolution – Knowledge Representation - Ontological Engineering-Categories and Objects – Events - Mental Events and Mental Objects - Reasoning Systems for Categories - Reasoning with Default Information

NOTE: First order predicate logic, Unification, Forward Chaining, Backward Chaining, Resolution=> Refer class notes

I. Logic programming

- **Logic programming** is a programming paradigm based on formal logic.
- A program written in a logic programming language is a **set of sentences** that is expressed as facts and rules of some problem domain.
- Here the **facts are constructed by predicate logic and the problems are solved by inferring(i.e., reasoning) on that knowledge.**
- In short Logic programming is expressed as

  Algorithm= logic + control [where "Logic" represents a logic program and "Control" represents different theorem-proving strategies]
- Major logic programming language families include ***Prolog, answer set programming (ASP) and Datalog***
- Rules are written in the form of ***clauses.*** Clauses are written **"backwards"** .Syntax
  - ✓ **H :- B_1, ..., B_n=>** are read as : **H if B_1 and ... and B_n.**
  - ✓ **H** is called the ***head*** of the rule and **B_1, ..., B_n is** called the ***body***.
  - ✓ Facts are rules that **have no body**, and are written in the simplified form: **H.**
  - ✓ H, B_1, ..., B_n are all atomic formulae, these clauses are called **definite clauses or Horn clauses(one literal** must be **positive)**

PROLOG(PROgramming in LOGic)

- developed in 1972 by Alain Colmerauer
- Prolog is a language used for ***symbolic and logic-based computation***.
- It is a **declarative language** (nonprocedural or very high level, specify **what** problem you want to solve rather than **how** to solve it).
- is partly **like a database** but much more powerful since can **infer new facts**
- A Prolog interpreter can follow facts/rules and answer queries by sophisticated search.
- Many expert systems have been written in prolog
- Prolog programs are **set of definite clauses** written in a notation different from standard FOL
- Prolog is a **typeless language, which means you do not declare types**
- Each line of the prolog program must end with full stop (.)
- Syntax of Prolog

1. Terms
2. Predicates
3. Facts and Rules
4. Programs and Queries

1. Terms

- ✓ In Prolog, *all data*—including Prolog *programs*—are represented by Prolog **terms**.
- ✓ there are ***four kinds*** of term in Prolog: *Constants*(atoms(strings enclosed in single quotes +numbers), ***variables***, and ***complex terms*** (or structures).

a. Constants

- Sequences of letters, digits, or underscore "_" that start with lower case letters.
- e.g., Numbers : 1.001, 2, 3.03 , Strings enclosed in single quotes: 'Mary', '1.01' , 'string'

b. Variables

- Sequence of letters digits or underscore that start with an upper case letter or the underscore.
- e.g., ?- likes(john, X)
- Underscore by itself is the special "anonymous" variable. => means we don't care about its value
- E.g., wealthy(employee(_, _, _, Salary)) :-Salary >= 100000.

c. Structure

- A structure in Prolog is a single object which consists of a collection of other objects, called components
- A structure can be decomposed into
  - A functor ; name of the structure
  - one or more components
- A structure has the form: ***structure-name (attribute,..., attribute)***
- E.g. To create an employee data structure : **employee(1234, 'Jones', 'James', 100000)**

d. Lists

- is a collection **of terms**
- Syntax : [t1,t2,...,tn]
- list elements are **enclosed by brackets and separated by commas**.
- In list , **direct access is only to the first element** called the **Head**, while the **rest forms the list** called the **Tail.**
- [Head|Tail] where **Head is a single element,** while **Tail is a list**
- List properties
  - ✓ list can have as **many elements** as necessary.
  - ✓ A list can be **empty;** an empty list is denoted as **[ ]**.

- ✓ A list can have arguments being of: 1) mixed types, 2) complex structures
- ✓ a list can have nested lists (to an arbitrary depth)

- operators for list

i) To split the list: **pipe** (|).

- ✓ [Head|Tail] = [red, white, black, yellow], Output : Head = red ,Tail = [white, black, yellow]
- ✓ [H1, H2|T] = [red, white, black, yellow]. Output: H1 = red H2 = white T = [black, yellow]

ii) To concatenate the list =>**append**

- ✓ append ([],L,L) => an empty list is append with list L and produces the same list L . E.g., append([a,b], [c,d,e], [a,b,c,d,e]).
- ✓ append([H|T],L,[H|TL]) :-append(T,L,TL). => This is a recursive definition
- ✓ Input:[H| T]+ L] Result: [H | TL]T+L

2. Predicates

- ✓ Predicates are syntactically identical to structured terms
  <identifier>($Term_1$, ..., $Term_k$)
- ✓ E.g., elephant(mary), likes(john, fred)

3. Facts and Rules

Facts

- ✓ Facts are **predicates** followed by a dot
- ✓ Facts are used to define something that is always true.
- ✓ e.g., likes(john , chicken)

Rules

- ✓ Rules consist of a **head** and a **body** separated by **:- (called as if)**
- ✓ Syntax: $predicate_H$:- $predicate_1$, ..., $predicate_k$. => The head of a rule is true if all predicates in the body can be proved to be true
- ✓ E.g., criminal(x) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).

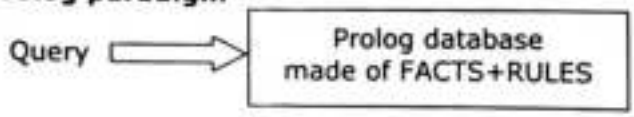
4. Program and queries

- ✓ **Programs**: Facts and rules are called clauses. A Prolog program is a **list of clauses**.
- ✓ **Queries**
  - are predicates (or sequences of predicates) followed by a dot.
  - They are typed in at the Prolog prompt
  - cause the system to reply
    - look at the database
    - try to find a match for the query pattern
    - execute the body of the matching head
    - return an answer

- Types of queries

1. **Ground Query**
   - No variable in the argument
   - E.g., ?- man (marcus)
   - Answer to the ground queries are Yes or No
2. **Non-ground query**
   - Atleast one argument is a variable
   - e.g., ?- father(F,C)
   - Answer to this queries are the substation values to the variables in the query
   - Done based on DFS

✓ **Prolog paradigm**



E.g. Conversion of FOL to Prolog

Logic	Prolog representation of logic
∀ x pet (x) ∧ small (x) ==> apartment (x)	apartment (x) :- pet (x) , small (x).
∀ x cat (x) ∨ dog (x) ==> pet (x)	pet (x) :- cat (x).
	pet (x) :- dog (x).
∀ x poodle (x) ==> dog(x) ∧ small(x)	dog(x) :- poodle (x).
	small(x) :- poodle (x).
poodle(fluffy)	poodle(fluffy).

E.g., **simple Prolog program (family.pl)**

```
male(albert).                    %a fact stating albert is a male
male(edward).
female(alice).                   %a fact stating alice is a female
female(victoria).
parent(albert,edward).           %a fact: albert is parent of edward
parent(victoria,edward).
father(X,Y) :-                   %a rule: X is father of Y if X if a male parent of Y
        parent(X,Y), male(X). %body of above rule. can be on same line.
mother(X,Y) :- parent(X,Y), female(X).  %a similar rule for X being mother of Y
```

To run this program, prolog interpreters must be used. Here **SWI-Prolog interpreter**

```
skywolf:~% prolog
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.6.58)
Copyright (c) 1990-2008 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.
For help, use ?- help(Topic). or ?- apropos(Word).

?-   ← this is the prompt. You can load you program and ask queries.

?- consult(family).    %loading our file. We can use full-name with quotation 'family.pl'

% family compiled 0.00 sec, 2,552 bytes    % this is the result of loading
true.                                       % true means loading file was successful

                    % alternatively we could have used ['family.pl']. to load our file.

?- halt.            %exiting SWI

skywolf:~%
```

ask queries after loading our program

```
?-male(albert).
Yes.                    %the above was true
?-male(victoria).
No.                     %the above was false

?-male(mycat).
No.

?-male(X).              %X is a variable, we are asking "who is male?"
X=albert;               %right! now type semicolon to ask for more answers.
X=edward;
No                      %no more answers

?-father(F,C).          %F & C are variables, we are asking "who is father of
whom"
F=albert, C=edward;     %this is awesome! How did Prolog get this?
No
```

Unification in Prolog

- ✓ The way in which Prolog matches two terms is called **unification.**
- ✓ If the two terms are structures

  t = f(t1, t2, ..., tk)

  s = g(s1, s2, ..., sn)

  Then these two terms match if

  - ➢ the predicates "f" and "g" are identical.
  - ➢ They both have same number of attributes
  - ➢ Each of the terms ti, si match (recursively).
- ✓ =(unifiability/ infix predicate) is used to unify two terms

S.No	Example	Description
1.	?- fred = fred. Ans: true.	fred unifies with fred - they are the same atom.
2.	?- X = fred. Ans: X = fred	X unifies with fred by binding X to fred.
3.	?- X = likes(mary, pizza).	X unifies with likes(mary, pizza) by binding X to

	Ans: X = likes(mary, pizza).	likes(mary, pizza).
4.	**Facts :** likes(john, jane). likes(jane, john). **Query :** ?- likes(john, X). Answer : X = jane.	asking the query first prolog start to search matching terms in 'Facts' in top-down manner for 'likes' predicate with two arguments and it can match likes(john, ...) i.e. Unification. Then it looks for the value of X asked in query and it returns answer X = jane i.e. Instantiation - X is instantiated to 'jane'.

Application of Prolog

1) Intelligent Data base Retrieval
2) NLP
3) Specification Language
4) Machine Learning
5) Robot planning
6) Automated reasoning
7) Problem solving

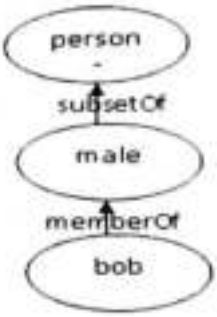
II. Knowledge Representation

2.1. Ontological Engineering

- **Ontology:** ***organizes*** everything in the world into a ***hierarchy of categories***
- **Categories :** acts as the ***primary building blocks in a large-scale for knowledge representation***
- Ontology Engineering: represents the abstract concept in KR
  - Events, time, physical objects that occur in different domains
- **Upper Ontology**: the convention of **drawing graphs** with the **general concepts** at the **top** and the more **specific concepts below them**

2.2. CATEGORIES AND OBJECTS

- KR organizes objects into categories
  - ✓ **interaction** with the world takes places at the **level of object**
  - ✓ **reasoning** takes places at the **level of categories**
- Categories plays a vital role in **prediction about objects**=> based on perceived properties
- Can be represented in **two ways** by FOL
  - ✓ predicate: **apple(x)**
  - ✓ Reification of categories as objects: **apples**
- Network that shows relations among categories is called **as taxonomy or taxonomic hierarchy or class/subclass relationships**



2.2.1 FOL and Categories

- **FOL=>** makes easy to state facts about categories, either by relating objects to categories or by quantifying over their members
- **Relation between categories**

1. An **object** is a **member** of a **category** => E.g., MemberOf(B12,Basketballs)
2. A **category** is a **subclass** of another **category =>** E.g., SubsetOf(Basketballs,Balls)
3. All **members** of a **category** have some **properties**
   E.g., ∀ x (MemberOf(x,Basketballs) ⇒ Round(x))
4. All **members** of a **category** can be recognized by **some properties**
   E.g., ∀ x (Orange(x) ∧ Round(x) ∧ Diameter(x)=9.5 ∧ MemberOf(x,Balls) ⇒ MemberOf(x, BasketBalls))
5. A **category** as a whole has some **properties**
   E.g., MemberOf(Dogs, DomesticatedSpecies)
6. Two or more categories are **disjoint** if they have no members in common
   - ✓ Disjoint(s)⇔(∀ c1,c2 c1 ∈ s ∧ c2 ∈ s ∧ c1 ≠ c2 ⇒ Intersection(c1,c2) ={})
   - ✓ Example; Disjoint({animals ,vegetables})
7. A set of categories s constitutes an **exhaustive decomposition** of a category c if all members of the set c are covered by categories in s:
   - ✓ E.D.(s,c) ⇔ (∀ i, i ∈ c ⇒ ∃ c2 c2 ∈ s ∧ i ∈ c2)
   - ✓ E.g., E.D ({Americans, Canadian,Mexicans}, NorthAmericans)
8. A **partition** is a **disjoint exhaustive decomposition**
   - ✓ Partition(s,c) ⇔ Disjoint(s) ∧ E.D.(s,c)
   - ✓ Example: Partition({Males,Females},Persons)

2.2.2 Natural Kind

- Many categories have **no clear-cut definitions** (chair, bush, book)
- Tomatoes: sometimes green, red, yellow, black. Mostly round.
- One solution: category **Typical**(Tomatoes).
- E.g., ∀ x, x ∈ Typical(Tomatoes) ⇒ Red(x) ∧ Spherical(x).

2.2.3 Physical Composition

- One object may be part of another: e.g., Chapter 10 is a part of AI Book
- Use part of predicate=> part Of(chapter10, AI Book)

- The PartOf predicate is **transitive and reflexive**
    a. ∀ x PartOf(x,x)
    b. ∀ x,y,z PartOf(x,y) ∧ PartOf(y,z) ⇒ PartOf(x,z)

2.2.4 Measurements

- Objects in the world have properties (i.e., attributes). The **values assigned** for these **properties** are called **measures**

2.3. EVENTS

- Facts are treated as true , independent of time
- Defn: describe **what is true**, when something is happening
- Can occur in 2 ways

1. Situation Calculus

- Represents **actions and effects**
- E.g., filling a bathtub :Tub is empty before action and full when the action is done
- Disadvantages
    - Doesn't talk about what happens during the action
    - Can't describe two action happing at a time(e.g., brush teeth while waiting for the tub to fill)

2. Event Calculus

- Based on **time rather than situation**
- Reifies (meaning is make something more real) Fluents and events
- Defn: Event calculus is **reasoning about change** / how **actions change the world**
- they were 3 SORTS of arguments
    1. e => set of events
    2. f => set of fluent(meaning , A fluent is a predicate, which can change essentially. e.g., dirt(x))
    3. T => time
- Complete **predicate of event calculus**
    1. T(f, t) : Fluent f is true at time t
    2. Initiates(e, f, t) : Event e causes fluent f to start to hold at time t/ is true, if event e makes fluent f true at time t
    3. Terminates(e, f, t) : Event e causes fluent f to cease to hold at time t
    4. Happens(e, i) : Event e happens over the time interval i
    5. Clipped(f, i): Fluent f is no longer true at time i
    6. Restored (f, i): Fluent f becomes true sometime during i

2.4. Mental Events and Mental Objects

- agents that constructed so far have beliefs and can deduce new beliefs.
- Yet none of them has any knowledge *about beliefs or about deduction.*

- *Knowledge about one's own* knowledge and reasoning processes is useful for controlling inference.=> the agent should **know what is knows and what it does not know**

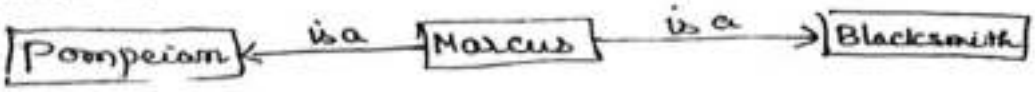
2.5. Reasoning Systems for Categories

- How to **organize and reason with categories**

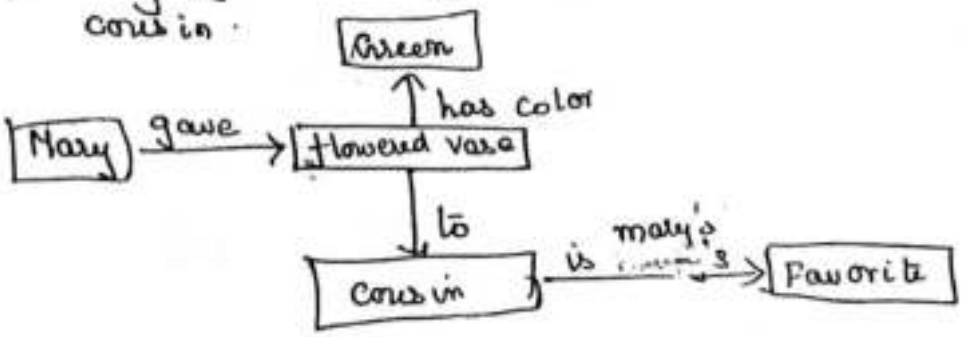
1. Semantic networks

- Represent knowledge in the form of **graphical networks**.
- Developed based on human memory model, applied in neural networks and expert system
- Semantic networks can categorize the object in different forms and can also link those objects
  - nodes =>represents objects and attributes values
  - arcs => describe the relationship between those objects/ attributes
- Uses efficient algorithms for inferring properties of an object on the basis of its category membership
- E.g.,

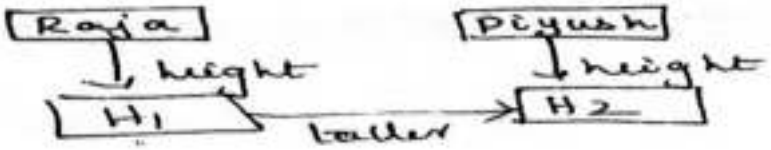
(a) Pompeian (Marcus), Blacksmith (Marcus)
Class



(b) Mary gave the green flowered base to her favorite cousin.

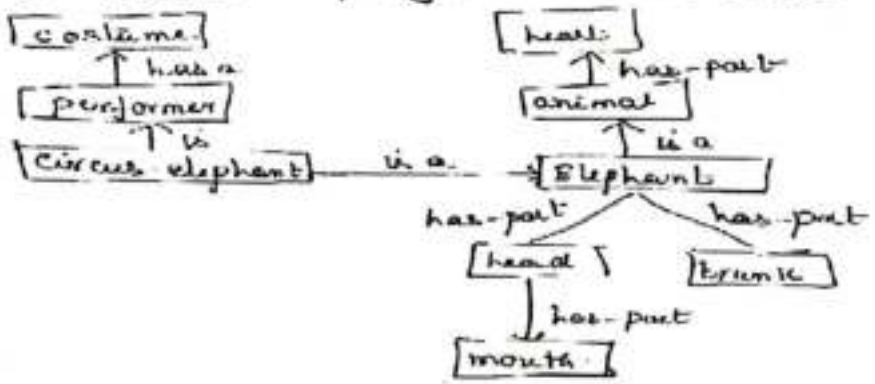


(c) Raja is taller than Piyush



Represent the following information in a semantic net

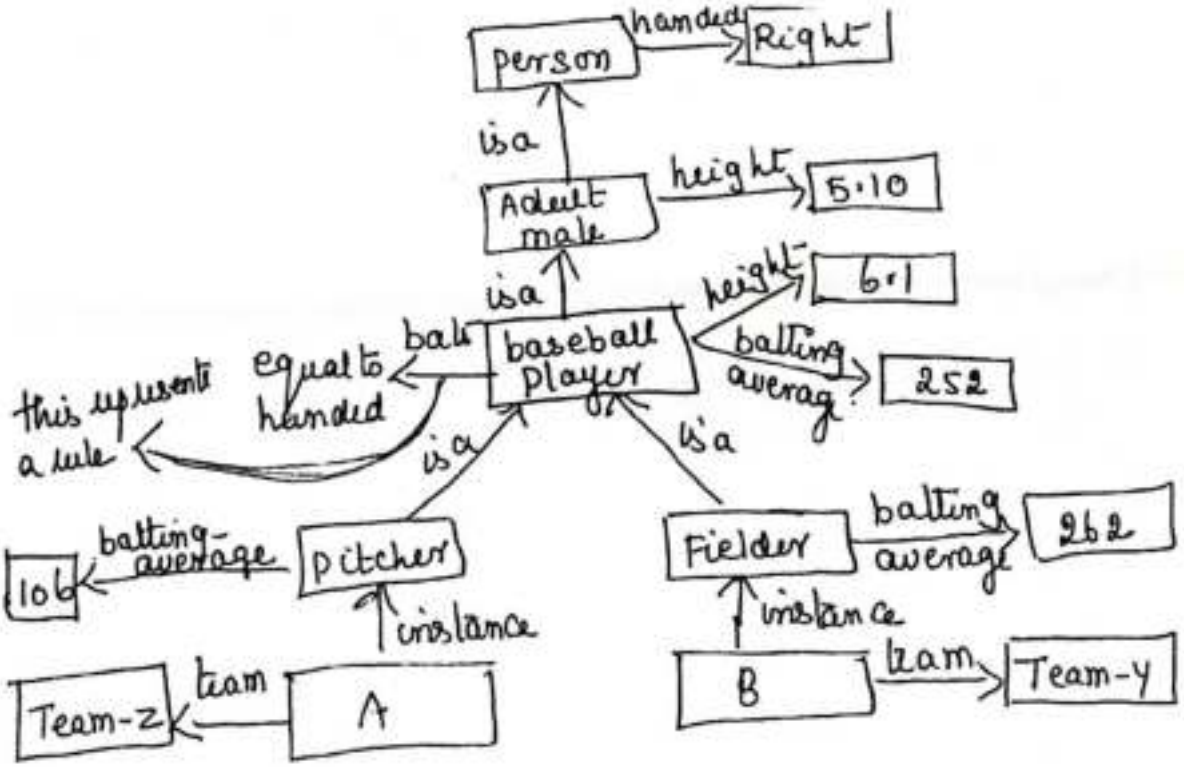
(i) isa circus-elephant elephant
(ii) has part elephant head
(iii) has part elephant trunk
(iv) has part head mouth
(v) isa elephant animal
(vi) has part animal heart
(vii) isa circus elephant performer
(viii) has a performer costume



- Basic inference mechanism: *follow* ***links between nodes.***

  i) inheritance reasoning ii) intersection search

  Consider the following semantic network



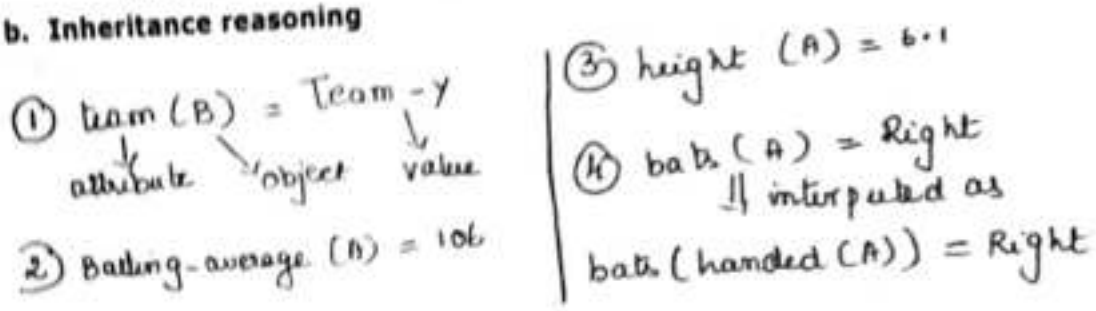
a. Intersection search

- Earliest way to **find relationships between objects** by **spreading activation** from each of two nodes and seeing where the **activations met**. This **process is called intersection search.**
- E.g.,

  Question: "What is the relation between Team-z and Team-y?

  Answer: "They are both teams of baseball players."

b. Inheritance reasoning



2. Description logic

- A logic based knowledge representation language
  - description" about the world in terms of concepts(classes), roles(properties, relationships) and individuals(instances)
  - Used in databases(e.g., DL CLASSIC) and semantic network(e.g, OWL language)
- Principal inference task is
  - **Subsumption**: checking if one category is the subset of another by comparing their definitions
  - **Classification**: checking whether an object belongs to a category.
  - **Consistency:** whether the category membership criteria are logically satisfiable

2.6. Reasoning with default information

- very common form of **non-monotonic reasoning**
- Logic will be said as non-monotonic if **some conclusions can be invalidated** by **adding more knowledge** into **our knowledge base.**
- **Example:** suppose the knowledge base contains the following knowledge:
  - **Birds can fly**
  - **Penguins cannot fly**
  - **Pitty is a bird** , we can conclude that **Pitty can fly**.
  - However, if we add one another sentence into knowledge base "**Pitty is a penguin**", which concludes "**Pitty cannot fly**", so it invalidates the above conclusion.

Initial call is

MINIMAX_A_B (current, 0, player-one, maximum value
STATIC can compute (+∞), minimum value
STATIC can compute (-∞))

MINIMAX_A_B (position, depth, player, use_threesh, pass_threesh)

Knowledge Representation:

→ Knowledge Representation are two distinct entities

knowledge: a description of world

Representation: the way knowledge is encoded

KR: definition

(Dedicated to represent information about the world in a form that a computer system can utilize to solve complex tasks.)

[Knowledge is the information about a domain that can be used to solve problems in that domain. To solve many problems requires much knowledge, & this knowledge must be represented in the computer. As a part of designing a program to solve problems, we must define how the knowledge will be represented. A representation scheme is the form of the knowledge that is used in an Agent. A representation of some piece of knowledge is the internal representation of the knowledge. A representation scheme specifies the form of the knowledge. A knowledge base is the representation of all of the knowledge that is stored by an Agent].

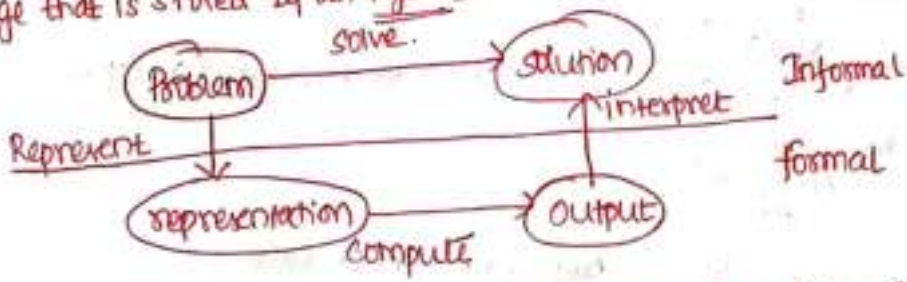


Fig The Role of Representations in solving Problems.

Representations and Mappings

(information, truth, data)

→ Knowledge is a collection of "facts" from some domain. Some representation are needed for facts to be manipulate in the program.

→ English cannot be used directly to draw inferences. Some symbolic representation is needed.

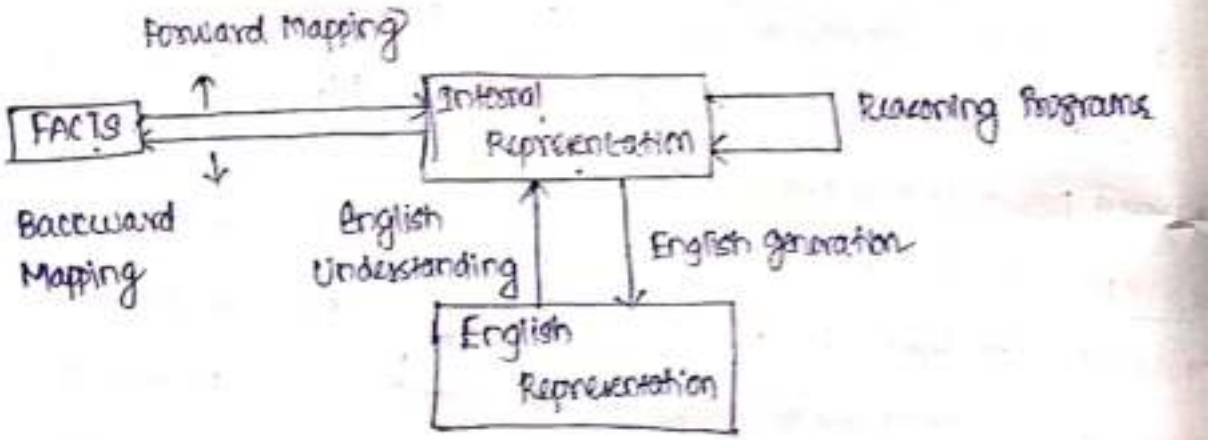
∴ Mapping facts to symbols (called as Representation)

symbols to facts (Forward and Backward mappings)

→ Facts are structured at knowledge level [place where facts are described]

Representations are structured at symbol level. [place where facts are described as symbols]

Pictorial Representation of Mapping between Facts & Representation



Eg:

FACT: Jimmy is a DOG.

Internal Representation : DOG(JIMMY) → done using Forward mapping.

(name) (specific)

DOG → Attribute/class, JIMMY → object

Here we assume logical representation of fact

FACT: ALL DOGS HAVE TAIL (internally).

∀ - For all

∃ - For someone / there exist

∨ - OR (~~Conjunction~~) Disjunction

∧ - AND (~~Disjunction~~) Conjunction

¬ - Negation / NOT.

→ . Implication.

↔ Bimplication.

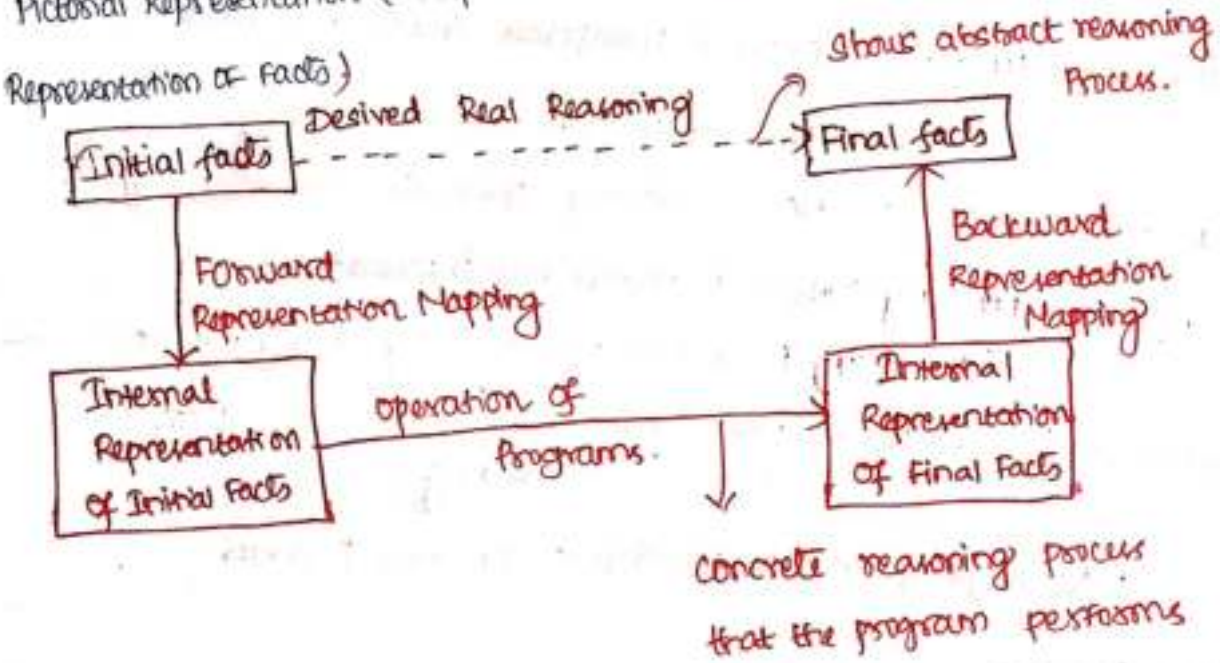
$\forall(x): dog(x) \rightarrow havetail(x)$

Using the above two logical representation, new knowledge is infered, and its logic form is

New knowledge; JIMMY HAS TAIL → done using Backward mapping.

logic Form: Tail (JIMMY)

Pictorial Representation (Expanded view of previous picture which shows Representation of facts)



Facts

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

Representation of the facts

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

Approaches to knowledge Representation

Properties that a knowledge representation system possess are

1. Representational Adequacy (capability)

The ability to represent all kinds of knowledge that are needed in that domain

(or)

should allow to represent the knowledge we need.

2. Inferential Adequacy (Ability to infer new knowledge)

The ability to manipulate the knowledge represented to produce new knowledge corresponding to that inferred from the original.

3. Inferential Efficiency (ability to incorporate Additional Information)

The ability to direct the inferential mechanisms into the most productive directions by storing appropriate guides.

4. Acquisitional Efficiency (ability to acquire new information)

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

∴ No single system that optimizes all the above properties

Types of knowledge

1. Simple Relational knowledge
2. Inheritable knowledge
3. Inferential knowledge
4. Procedural knowledge
5. ~~Declar~~ Declarative knowledge.

1. Simple Relational knowledge

The Facts are represented as set of relations in a tabular form. knowledge representation in this form act as input to inference engine. It is very simple.

The main disadvantage is it provides weak inferential capabilities (i.e it cannot infer new knowledge)

Eg: knowledge regarding students

Name	height	Weight	Batting
A	6.0	180	Left
B	5.10	170	Right
C	6.2	215	Left
D	6.3	205	Left

This type of representation use to answer simple question

Eg: "Who is the tallest boy?, but it cannot answer questions/queries like

Eg: "A' is a good boy"

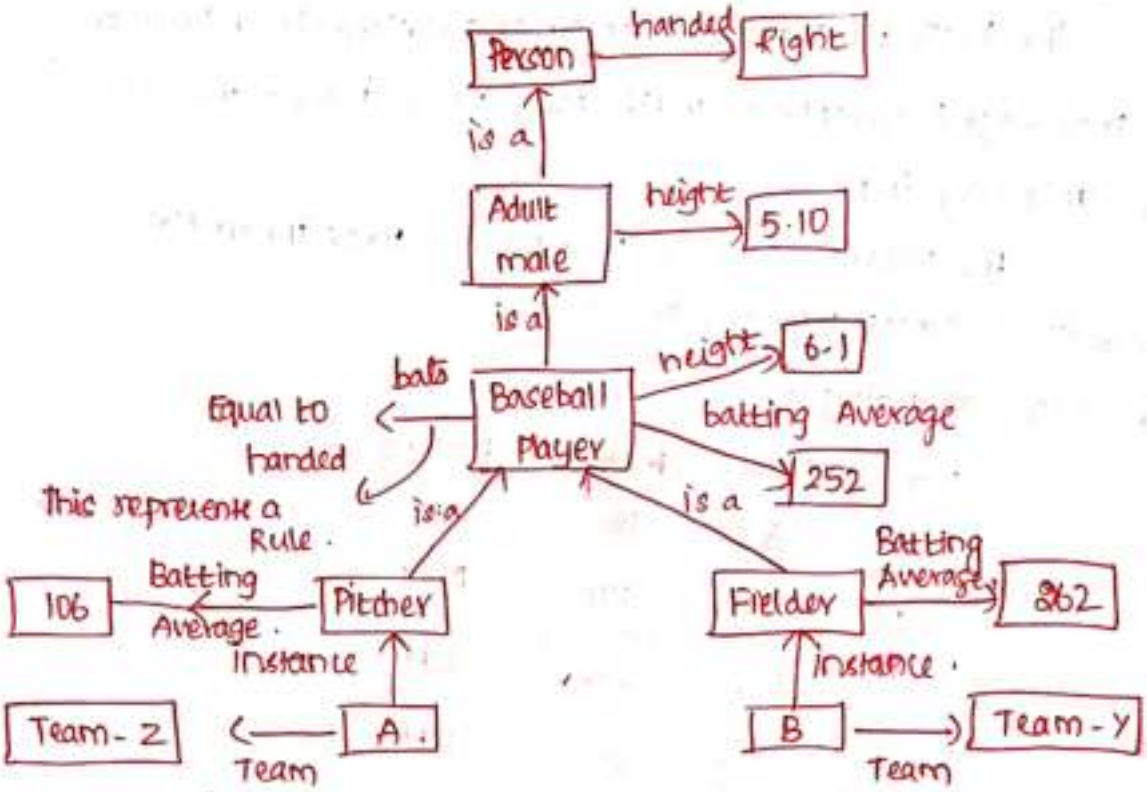
2. Inheritable knowledge

Relational knowledge represents facts as set of attributes and its values, but does not support inference mechanism.

Agumenting Basic representation with inference mechanisms to operate on the structure of representation [called as property inheritance => in which elements of specific classes inherit attributes and from general classes].

To support property inheritance, Objects are organized into classes and classes are arranged in a generalization hierarchy.

Pictorial Representation of Baseball player in the form of inheritable knowledge.



Property Inheritance

Elements inherit values from being members of a class.

Data must be organized into a hierarchy of classes.

Boxed nodes → Objects and values of attributes of objects

Values can be objects with attributes and so on.

Arrows → point from object to its value.

This structure is known as slot and filler structure, semantic network or a collection of frames.

3. Inferential knowledge

Represents knowledge as formal logic (First order logic)

Generates new information from the given facts.

Logic provides a powerful mechanism to describe relationships among values.

Eg:

Jimmy(dog)

$\forall x$: dog(x) → hastail(x)

⇓ Inferred

hastail(Jimmy)

4. Procedural knowledge / operational knowledge / Imperative knowledge.

[Implicit → you are no longer consciously aware of the knowledge]

Basic Idea:

Knowledge encoded in some procedures (small Pgms that know how to do specific things, how to proceed).

Specifies what to do when (ie specifies when to use knowledge)

It can be (procedural knowledge) can be represented as.

1. Production Rules [Is knowing how to do something].
2. Writing a code in a language like LISP.

5. Declarative knowledge (DL) [Explicit → You know that you know it]

[Is knowing about something]

- Representation of facts or assertion in abstract
- Specifies knowledge, but does not specify where the knowledge is used. Frame representation is used to represent DL.

Key features / categories → Facts, world or personal history.

Eg: How to cook vegetable or how to prepare a particular dish is Procedural knowledge.

Eg:

The first step in cooking a vegetable is chopping it

To prepare a dish one needs a gather its ingredients

Procedural	Declarative.
1. Process of planting herbs	knowing something about trees.
2. Procedure of treating the crops of a particular disease	Description of symptoms of a plant disease
3. Procedure to harvest a crop	knowledge of the month when a crop should be harvested.
4. Draw a line graph from a data set	Familiarity with the data sets & line graphs
5. Procedure to register in a video lecture	knowledge acquired in a video lecture.

Knowledge Representation using Predicate Logic

Logic: It is the art of correct Reasoning

Predicate: Part of the sentence containing a verb and stating something about the subject.

Eg: went home in "John went home"

Well-formed formula:

A sentence in which all their variables are properly introduced with logical connectives (propositional) or quantifiers (predicate) → FOPL Ex

Facts are represented using logic (i.e symbolic language) because the logic is a powerful way to derive new knowledge from old facts (called as mathematical deduction)

Two ways to Represent

1. Propositional Logic / Boolean Algebra
2. Predicate logic

1. Propositional logic / Propositional calculus / sentential calculus.

→ Declarative statements that can be either true or false but not both.

Eg: It is raining → P
Raining

It is HOT → Q
HOT

It is windy → R
windy

↓ Conclusion

If it is raining, then it is not sunny

Raining → ¬sunny ⇒ P → ¬Q (called as WFF)

well formed formula.

Yes (or) No.

$\frac{7\times5}{35} + 4 \neq \frac{48}{39}$ → false

Two different statements can be combined to form a compound statement.

Sham is an honest boy. → P

Sham is hard working. → Q.

P ∧ Q.

Sham is an honest boy & hardworking

Limitations in propositional logic

1. Hard to represent statements about similar objects

   Eg: Socrates is a man => SOCRATESMAN
   Plato is a man => PLATOMAN
   } Unable to draw conclusion about similarities between socrates & plato

2. Hard to represent relationship.

   Eg: All men are mortal => MortalMan
   ⇓
   Fails to show relationship between any individual being a man and that individual being a mortal.

To avoid these two limitations the statement must be written (i.e represented) with variables and quantifiers. [can be done using predicate logic]

2. Using Predicate logic / Predicate calculus. [To remove the disadvantage of Propositional]

   → A predicate is a proposition whose truth depends on the value of one or more variables. // Extension of propositional logic.

   * Predicate logic uses variables and quantifiers to represent facts
   * Quantifiers used for expressing properties of entire collection of objects, rather that just a sing object

& Types

i) Universal
   ∀ (symbol) => Pronounced as "For All"

ii) Existential
   ∃ (symbol) => Pronounced as "There exist" or "For some"
   Used to make statements about some objects and not just for all.

Eg: (Predicate logic).

1. Fact: Marcus was a man

WFF: man(marcus) [In predicate logic].

2. FACT: Marcus was a pompeian

Pompeian(Marcus)

{ ∧ - conjunction
  ∨ - disjunction

3. All pompeian were Romans

∀(x): pompeian(x) ⟶ Roman(x)

4. Caesar was a Ruler:

Ruler(caesar)

5. All romans were either loyal to caesar or hated him

∀(x): Romans(x) ⟶ loyalto(x, caesar) ∨ hate(x, caesar)

∨ ↓ Disjunction.

6. Everyone is loyal to someone

∀(x): ∃y: loyalto(x, y)

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

∀x: ∀y: people(x) ∧ ruler(y) ∧ tryassinate(x, y) ⟶ ¬ loyalto(x, y)

8. Marcus tried to assassinate caesar

tryassassinate(Marcus, caesar)

(1) (2)

---

1. Sky is blue (Two terms only sky & blue) (M) knife is weapon & we always know that sky is blue, so represent in implication.

Sky → blue.

↳ Implication.

2. John is a king. (It cannot be represented in Implication, since John cannot be alone a king).

king (John)
↓ class ( ) object

3. Everyone likes Icecream.

∀(x) likes (x, icecream)

John like Icecream

likes (John, Icecream)

Consider the Following Sentences into predicate logic

a) John likes all kinds of Foods → everyone ∀ ∀(x)

b) Apples are food

c) Chicken is Food

d) Anything (que) Anyone (quantifier) eats and isn't killed by his Food
↳ Represent Food Item.

e) Bill eats peanuts & is still alive

f) Sue eats everything Bill eats

Answer

1. ∀x Food(x) → likes (John, x)
2. Apple(x) → Food(x) class(object) → food(Apple)
3. Chicken(x) → Food(x) → Food(chicken)
4. ∀x : ∀y eats(x, y) ∧ ¬ ¬killed(y) → Food(x)
5. eats(Bill, peanuts) ∧ ¬killed(Bill)
6. ∀x eat(Bill, x) → eat(sue, x).

Eg:1 The Ball's colour is Red. [In propositional we will represent in a single term say P or Q or R].

Fact in predicate ②.

Color (Ball, Red)
② ①
↓ ↓
Predicate Arguments

Eg:2

Rohan likes Bananas.

likes (Rohan, Bananas)

↓ Predicate

Arguments, Objects.

Eg:3

All students are Intelligent

Rohan is a student.

Inference → Rohan is Intelligent

Eg. 4

Everybody loves somebody.

All

∨, ∧, ¬, →, ↔ (propositional)

For all, someone (No symbols)

Quantifiers introduced in Predicate logic

↓

To express English words including All & some

Universal Quantifier → ∀ (For All)

Existential Quantifier → ∃ (There Exist)

Everybody loves somebody.

All

There Exist

Keep one variable for Everybody & one variable for y.

Loves (Everybody, somebody)

∀x ∃y. L(x, y)

So the total conclusion is ∀x ∃y L(x, y)

Representing instance and isa Relationships

→ Used in Property Inheritance reasoning

→ which captures the relationship such as class membership and class inclusion.

3 ways to Represent

1. Marcus was a man
2. Marcus was a pompeian
3. All pompeians were Roman
4. Caesar was a ruler
5. All romans were either loyal to caesar or hated him.

a) Direct predicate Representation

1. man (marcus)
   - man → class
   - marcus → Object
2. pompeian(marcus)
3. $\forall x$: pompeian(x) $\rightarrow$ Roman(x)
4. ruler (Caesar)
5. $\forall x$: Roman(x) $\rightarrow$ loyalto(x, caesar) $\vee$ hate(x, caesar)

b) Using Instance predicate

Syntax (x, y)

Instance (x, y) → class to which the object belongs to

↓

Object

1. Instance(marcus, man)
2. Instance (pompeian, marcus)
3. $\forall x$: Instance (x, pompeian) $\rightarrow$ Instance (x, Roman) [superclass – Roman subclass – pompeian]
4. Instance (caesar, ruler)
5. $\forall x$: Instance(x, Roman) $\rightarrow$ loyal to (x, caesar) $\vee$ hate (x, caesar)

c) Is a

3. Is a (pompeian, Roman).

Computable Functions and Predicates

Computable Predicate

→ Transformation of object whose value are true or false

eg: gt(1,0), gt(2,3)

Computable Function

An algorithm expects some input, does some calculation and if it terminates, returns a unique result.

eg: gt(2+3, 6)

↳ Evaluate this first [2+3=5], then call gt function with 5,6.

Eg:

1. Marcus was a man

   man(marcus)

2. Marcus was born in 40 A.D

   born(marcus, 40)

3. Marcus was a pompeian

   Pompeian(marcus)

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

   erupted(volcano) ∧ ∀x: [pompeian(x) ⟶ died(x, 79)]

5. All men are mortal

   ∀x: man(x) ⟶ mortal(x)

No mortal lives longer than 150 yrs.

∀x: ∀t1: ∀t2: mortal(x) ∧ born(x, t1) ∧ gt(t2 − t1, 150) ⟶ died(x, t2)

↓

Computable Function

Resolution (Resolve something)

→ Produces proof by Refutation/contradiction (i.e Negation)

→ To prove a statement, resolution attempts to show that the negation of the statement, produces a contradiction with the known statement (i.e it is unsatisfiable).

→ Resolution can be applicable for the facts that are represented in clause form.

∴ A clause is a special formula as disjunction of literals.

i) If a clause contains only one literal it is called as 'unit clause'

ii) A clause having no variable it is called as "ground clause".

Steps for Resolution

1. Convert the given statements in predicate / propositional logic.
2. Convert these statements into conjunctive Normal Form
3. Negate the conclusion (proof by contradiction)
4. Resolve using a Resolution Tree (Unification)

Steps to convert to CNF / clause form

Every sentence in propositional logic is logically equivalent to a conjunction of disjunction of literals. A sentence expressed as a conjunction of disjunctions of literals is said to be in conjunctive Normal Form (or) CNF.

1. Eliminate implication '→'

$a \rightarrow b = \neg a \vee b$

$\neg(a \wedge b) = \neg a \vee \neg b$ } Demorgan's law

$\neg(a \vee b) = \neg a \wedge \neg b$

$\neg(\neg a) = a$

2. Eliminate Existential Quantifier '∃'

To eliminate an independent Existential Quantifier, replace the variable by a skolem constant. This process is called as skolemization.

Eg: ∃y: President(y)

Here 'y' is an independent Quantifier so we can replace 'y' by any name (say- Bush)

so, it becomes president(Bush)

3. Eliminate Universal Quantifier '∀'

To eliminate the universal quantifier, drop the prefix in PRENEX NORMAL FORM i.e just drop ∀ and the sentence then becomes in PRENEX NORMAL FORM.

4. Eliminate AND '∧':

a∧b split the entire clause into two separate clauses i.e a & b.

(a∨b)∧c → split the entire clause into two separate clauses a∨b and c.

(a∧b)∨c → split into two a∨c and b∨c.

To Eliminate ∧ break the clause into two, if you cannot break the clause, distribute the OR '∨' & then break the clause.

Steps to convert FOL to CNF

WFF (Well Formed Formula)

1. Eliminate Biconditional (⟷)

$a \leftrightarrow b \Rightarrow (a \rightarrow b) \wedge (b \rightarrow a)$

$(\neg a \vee b) \wedge (\neg b \vee a)$

2. Eliminate Implication (→)

$a \rightarrow b \Rightarrow \neg a \vee b$

3. Move ¬ inwards

(Reduce scope of each ¬ to single term)

$\neg(a \wedge b) = \neg a \vee \neg b \ldots$

$\neg(a \vee b) = \neg a \wedge \neg b$

$\neg(\forall x\, a) = \exists x\, \neg a$

$\neg(\exists x a) = \forall x \neg a$.

$\neg\neg a = a$

4. Standardize Variables

Each Quantizer binds a unique variable.

$\forall x : P(x) \vee \forall x : Q(x)$

Convert to $\forall x : P(x) \vee \forall (y) : Q(y)$

5. Move all Quantifiers to the left of the Formula without changing their relative order. This is possible since there is no conflict among variable name. This is called Prenex Normal Form.

Move all Quantifier to front/left ↓, Consist of

Prefix of quantifiers followed by matrix (i e literals) with Quantifier Free.

Eg: $\forall x : P(x) \vee \forall y : Q(y)$

$\forall x \forall y : P(x) \vee Q(y)$

6. Eliminate existential Quantifier (helps to avoid assertion)

→ It is done by skolemization (is a process of replacing $\exists$ variables either by special constants known as skolem constants or by a special function known as skolem functions)

(or)

Eliminating the $\exists$ by substituting its variable a reference to a function that produces the desired value.

Eg: $\exists x : Rich(x)$

↓ x replaced by skolem constant G1

$Rich(G1)$

Eg2: $\exists x : President(x)$ ⟶ Transformed into $President(S1)$

↳ is a function with no arguments which produces a value that satisfies president called as skolem constant.

Eg2: If $\exists$ occurs within the scope of $\forall$, then the value satisfies the Predicate may depend on the value of the $\forall$ variable.

Eg: $\forall x : \exists y :$ Father-OF (y, x) => The value OF y that satisfies Father-OF depends on the Particular value of x.

Transformed ⇓

$\forall x :$ Father-OF $(S1(x), x)$

↓

Skolem Function.

6. 7. Drop Universal Quantifier ($\forall$) / Prefix

Eg: $\forall x :$ Person(x)

Converted → Person(x)

8. Convert the matrix (i.e literals) into a conjunction of disjuncts using (conjunction OF Disjunction) ($\wedge$ (AND) $\vee$ (OR))

a) Associative Property

$a \vee (b \vee c) = (a \vee b) \vee c$

b) Distributive Property

$(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)$

$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c) \rightarrow$ CNF

$2(x + y) = 2x + 2y.$

9. ~~Get~~ Create a separate clause corresponding to each conjunction

10. Standardize the variables in the set OF clauses generated in step 8.

Eg: Convert FOL to CNF

Def: A Formula is said to be Conjunctive Normal form if it consists in the conjunction of clauses $A_1 \wedge A_2 \wedge \ldots \wedge A_n$.

Where A is a clause.

Eg: $((P \rightarrow Q) \rightarrow R)$.

Step1: Eliminate Biconditional / Implication

$((P \rightarrow Q) \rightarrow R)$

$\Rightarrow ((\neg P \vee Q) \rightarrow R)$

$\Rightarrow \neg(\neg P \vee Q) \vee R.$

$a \rightarrow b = \neg a \vee b.$

$P \rightarrow Q = \neg P \vee Q.$

$(\neg P \vee Q) \rightarrow R.$

Step2: Move ¬ inwards.

(~~¬¬P∨¬R~~)

¬(¬P∨Q)∨R ⇒ ¬¬P∧¬Q∨R.

⇒ (P∧¬Q)∨R.

Step3: Apply Distributive law.

(a∧b)∨c = (a∨c)∧(b∨c)

(P∧¬Q)∨R = (P∨R)∧(¬Q∨R)
A_1 = A_2.

Eg:2

Convert the following statement in CNF using Predicate logic

Fact

Fact: Everyone who loves all animals is loved by someone.

Converting into Predicate logic: $\forall x : [\forall y : animal(y) \rightarrow loves(x,y)] \rightarrow \exists z : loves(z,x)$

Applying CNF

Resolution in Predicate logic example

Step1:

Eliminate biconditional

$\forall x : [\forall y : animal(y) \rightarrow loves(x,y)] \leftrightarrow \exists z : loves(z,x)$.

Based on biconditional $[a \rightarrow b \Rightarrow \neg a \vee b]$.

$\forall x : [\forall y : \underline{animal(y)} \vee loves(x,y)] \rightarrow \exists z : loves(z,x)$

The scope of y → is within the [ ] so negation in animal

$\forall x : \neg$

i) $\forall x : [\forall y : \neg animal(y) \vee loves(x,y)] \rightarrow \exists z : loves(z,x)$

↓Again there is biconditional

$\forall x : \neg [\forall y : \neg animal(y) \vee loves(x,y)] \vee \exists z : loves(z,x)$

Step2:

Move ¬ (Negation inward)

$\forall x : \exists y : animal(y) \wedge \neg loves(x,y) \vee \exists z : loves(z,x)$

Step3)

Standardize the variable of each quantifier

Here all the quantifier has unique variable.

Step4) Move all the quantifier to the left side / Prenex Normal Form

$\forall x : \exists y : \exists z$: animal (y) $\wedge$ $\neg$loves (x,y) $\vee$ loves (z,x)

Step5: Eliminate Existential quantifier / Skolemization

$\forall x$: animal ($f(x)$) $\wedge \neg$loves (x, $f(x)$) $\vee$ loves ($f_1(x)$, x)

↓ skolem function — ↳ skolem function

Step6: Drop the prefix / Universal quantifier.

[animal ($f(x)$) $\wedge \neg$loves (x, $f(x)$)] $\vee$ loves ($f_1(x)$, x)

Step7: Convert the literals into conjunction of disjunction

By Distributive property.

$(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)$

[animal ($f(x)$) $\vee$ loves ($f_1(x)$, x)] $\wedge$ [$\neg$loves (x, $f(x)$) $\vee$ loves ($f_1(x)$, x)]

Step8: Create separate clause corresponding to each conjunction

a) animal ($f(x)$) $\vee$ loves ($f_1(x)$, x)

b) $\neg$loves (x, $f(x)$) $\vee$ loves ($f_1(x)$, x)

Step9: Standardize the variable

a) animal ($f(x_1)$) $\vee$ loves ($f_1(x_1)$, x_1)

b) $\neg$loves (x_2, $f(x_2)$) $\vee$ loves ($f_1(x_2)$, x_2)

Eg:3

FACT: All Romans who know Marcus either hate Caesar or think that anyone who hates anyone is crazy.

Step1: Converting the given FACT into its equivalent predicate logic/WFF

$\forall x : [Roman(x) \wedge know(x, Marcus)] \rightarrow [hate(x, caesar) \vee (\forall y : \exists z : hate(y,z) \rightarrow thinkcrazy(x,y))]$

[Not everyone will be a enemy to a person, so someone. that someone is represented by $\exists$]

Step2: Predicate logic is formed, now convert the predicate logic to its equivalent CNF

Eliminate bicorditional / Implication

Predicate logic: $\forall x : [\underbrace{Roman(x) \wedge know(x, Marcus)}_{a}] \rightarrow [\underbrace{hate(x, caesar)}_{b} \vee (\forall y : \exists z : hate(y,z) \rightarrow thinkcrazy(x,y))]$

To remove implication $[a \rightarrow b \Rightarrow \neg a \vee b]$.

Now, the scope of x variable is defined for the overall logic,

So, $\forall x : \neg[Roman(x) \wedge know(x, marcus)] \vee [hate(x, caesar) \vee (\forall y : \exists z : \underbrace{hate(y,z)}_{a} \rightarrow \underbrace{thinkcrazy(x,y)}_{b})]$

Next, the second $a \rightarrow b$ will be

$\forall x : \neg[Roman(x) \wedge know(x, Marcus)] \vee [hate(x, caesar) \vee (\forall y : \neg(\exists z : hate(y,z)) \vee thinkcrazy(x,y))]$

The above logic implication is removed.

Step3: Move $\neg$ inwards (or) Reduce the scope of Negation [By Demorgan's law]

$\forall x : [\neg Roman(x) \vee \neg know(x, marcus)] \vee [hate(x, caesar) \vee (\forall y : \forall z : \neg hate(y,z) \vee thinkcrazy(x,y))]$

?

Step 4:

Standardize the Variable of each quantifier, binds a Unique Variable

From the WFF in step 3, ~~Each~~ All Universal quantifier has Unique Variable such as x, y and z.

Step 5:

Move all the quantifiers to the left of the formula without changing their relative order.

$\forall x : \forall y : \forall z : [\neg Roman(x) \vee \neg know(x, marcus)] \vee [hate(x, caesar) \vee (\neg hate(y, z) \vee thinkcrazy(x, y))]$

At this point, the formula is in what is known as Prenex Normal Form. It consists of a prefix of quantifiers followed by a matrix, which is quantifier-Free.

Step 6:

Eliminate Existential Quantifier / skolemization

From the ~~Fact~~ formula from step 5, their is no existential quantifier available.

Step 7:

Drop the prefix [i.e remove all Universal quantifier]

$[\neg Roman(x) \vee \neg know(x, marcus)] \vee [hate(x, caesar) \vee [\neg hate(y, z) \vee thinkcrazy(x, y))]$

Step 8: Convert the matrix (literals) into a conjunction of disjuncts.

In the case of our example, since there are no and's ($\wedge$) it is only necessary to exploit the associative property of or [i.e., $(a \wedge b) \vee c = (a \vee c) \wedge (b \wedge c)$] and simply remove the parentheses given,

$(a \wedge b) \wedge c = a \wedge (b \wedge c)$ [Associative]

¬ Roman(x) ∨ ¬know(x, Marcus) ∨ hate(x, Caesar) ∨ ¬hate(y,z)
∨ thinkcrazy(x,y)

Step 9: Create separate clause corresponding to each conjunction
NO conjunction (∧) operator available in our example, this is not applicable.

Step 10: Standardize the Variable.
Standardize apart the variable in the set of clauses generated in step 9. [Rename the variables so that no two clauses make reference to the same variable]

From step 8
⇒ ¬Roman(x_1) ∨ ¬know(x_1, marcus) ∨ hate(x_1, Caesar) ∨
¬hate(y_1, z_1) ∨ thinkcrazy(x_1, y_1) //.

The above WFF is known to be in conjunctive Normal Form (CNF).

Procedure For Resolution

The resolution procedure is a simple iterative process: at each step, two clauses, called the parent clauses, are compared (resolved), yielding a new clause that has been inferred from them.
The new clause represent ways that the two parent clauses interact with each other.

Eg: 1 Winter ∨ summer, ¬Winter ∨ cold. → Two Parent clause which contain same literal (eg Winter) one in +ve form and in other -ve form.

summer ∨ cold (Resolvent → It is obtained by combining all of the literals of two parent clauses except the ones that cancel)

Eg: 2 Winter, ¬Winter → If the clause that is produced is the empty clause then a contradiction has been found.

↳ Empty clause / NIL

It represents contradiction, If contradiction exists, the resolution Procedure terminates, otherwise, the procedure is not terminated.

Definition for Resolution:

It is a simple iterative process (two clauses) called Parent clauses are compared (resolved), yielding a new clause that has been inferred from them.

Resolution in Propositional logic

Eg:1

To Prove: ¬P.

given statement/Axioms are 1. P→Q

2. Q→R

3. ¬R

Step1:

Convert the above 3 statements in CNF form, Eliminate the biconditional by (a→b ⇒ ¬a∨b)

1. P→Q ⇒ ¬P∨Q

2. Q→R ⇒ ¬Q∨R

3. ¬R ⇒ ¬R

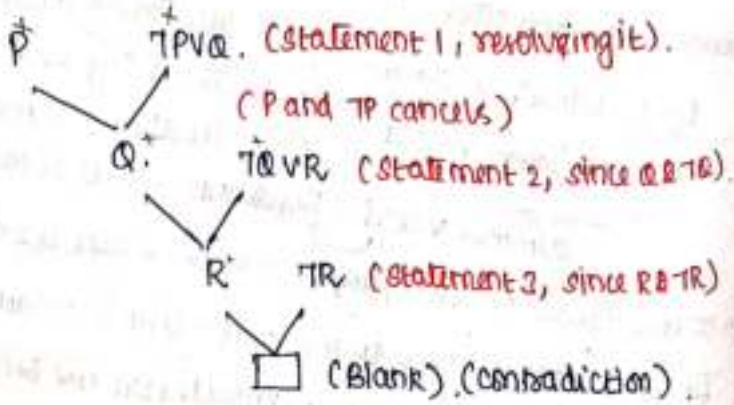
The above statements is in CNF form.

Step2: Proof for ¬P.

In this step Negate the proof to be proved (To prove: ¬P)

¬(¬P) = P.

Now check for the contradiction of above statement (P), the contradiction is in the first statement.



At the last, it is blank (i.e empty) it represents contradiction (Empty clause) it means that we cannot prove p & will can get only the negation of P. (i.e ¬P)

Eg:2

The procedure for producing a proof by resolution of Proposition 'P' with respect to a set of axioms F is.

Given Axioms are

1. P
2. $(P \wedge Q) \rightarrow R$ Prove that R
3. $(S \vee T) \rightarrow Q$
4. T

Step1: The given axioms are converted to its equivalent CNF/clause form.

1. $P \Rightarrow P$.
2. $(P \wedge Q) \rightarrow R$ [To eliminate Implication $(\rightarrow)$ $[a \rightarrow b \Rightarrow \neg a \vee b]$.

   i) $\neg(P \wedge Q) \vee R$.

   Moving $\neg$ inwards

   ii) $\neg P \vee \neg Q \vee R$ (CNF Form)

3. $(S \vee T) \rightarrow Q$ [Apply $a \rightarrow b \Rightarrow \neg a \vee b$]

   i) $\neg(S \vee T) \vee Q$.

   Moving $\neg$ inwards

   ii) $(\neg S \wedge \neg T) \vee Q$ [This can be solved using Distributive Property.

   $(a \wedge b) \vee c \Rightarrow (a \vee c) \wedge (b \vee c)$]

   ⇓

   (2) $(\neg S \vee Q) \wedge (\neg T \vee Q)$

   Separating the clause.

   a) $(\neg S \vee Q)$ b) $(\neg T \vee Q)$ (CNF Form)

4. $T \Rightarrow T$

∴ The CNF of above axioms are

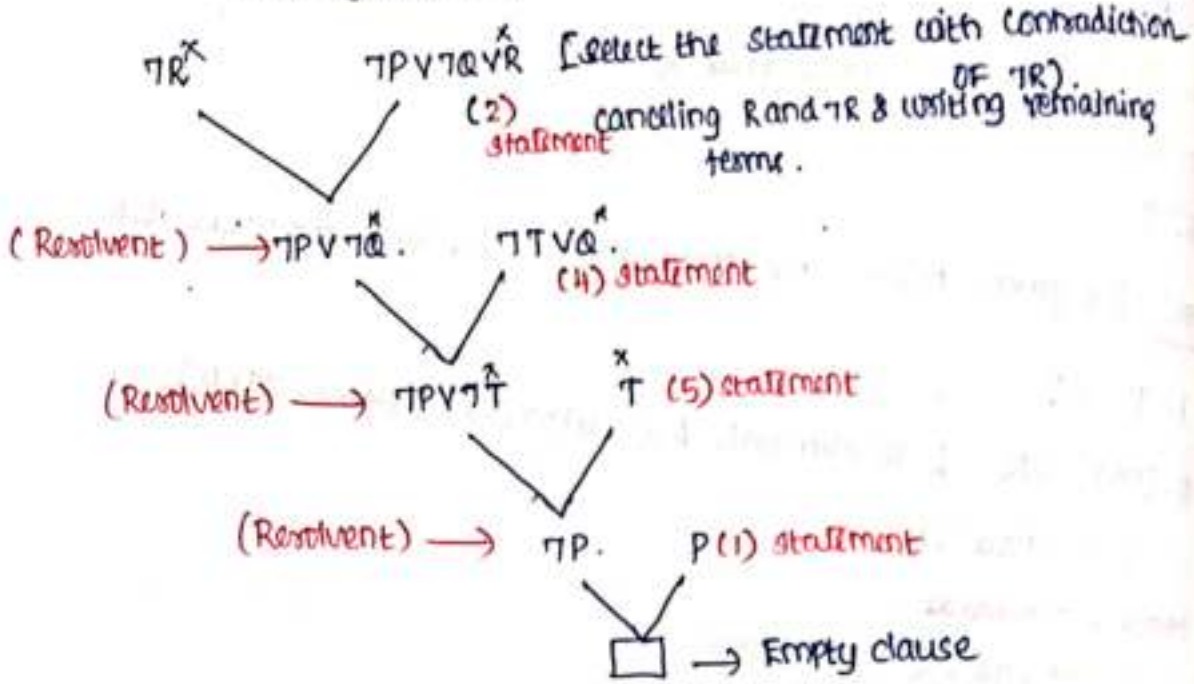
1. P
2. $\neg P \vee \neg Q \vee R$
3. $\neg S \vee Q$
4. $\neg T \vee Q$.
5. T

Step 2:

To Prove : R.

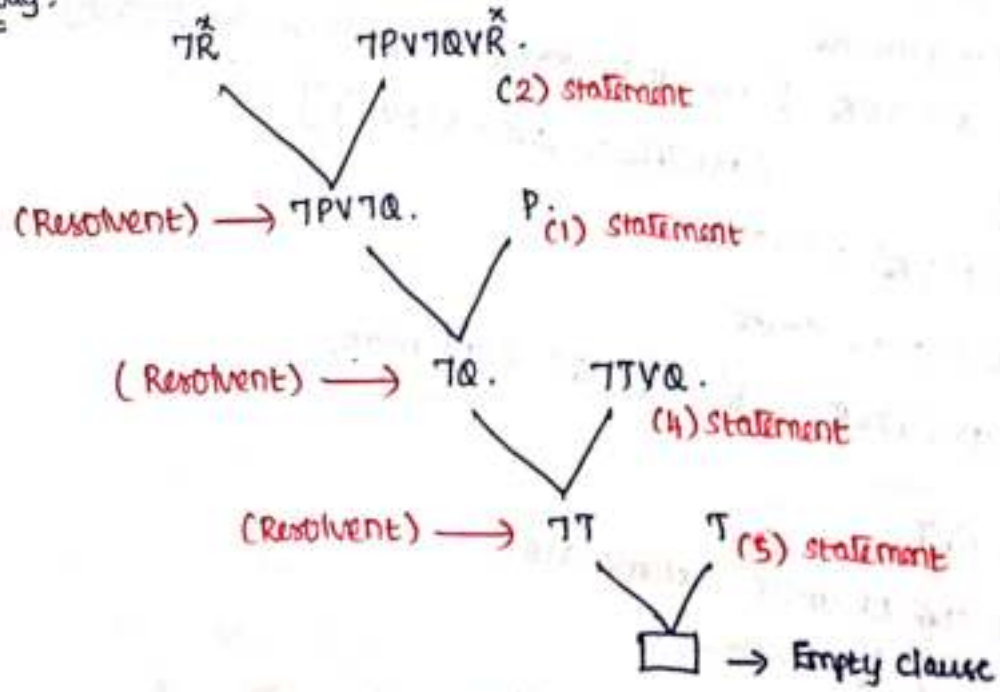
[To prove a statement, take Negation of the statement & prove it.

¬R. (Negating the statement to be proved)



It represent a Empty clause/contradiction, it means we cannot Prove ¬R and we can proof only the contradiction.

Other way:



Resolution in Predicate Logic

It uses Unification process.

Unification:

A matching procedure that compares two literals and discovers whether there exists a set of substitutions that make them identical. It is a recursive procedure.

Steps:

To unify two literals

1) check if their initial predicate symbols are same, if so we can proceed, otherwise say, cannot be unified.

Eg: Eryassassinate (Marcus, caesar)
Not same: hate (Marcus, caesar)

2) If the predicate symbols match, check the no. of arguments if equal proceed, otherwise say cannot be unified.

3) If the no. of arguments are equal, apply unification rules for each pair of argument at a time.

Eg: Eryassassinate (Marcus, caesar)
Different Predicate: hate (x, caesar) → Marcus/x
cannot be done using unification

Note: Unification Rules.

1) a variable can be replaced by a constant

eg: Marcus/x.

Eg: hate (marcus, caesar)
hate (x, caesar)
Predicate matches. (Marcus/x.)
Can done by Unification

2) a variable can be replaced by a variable

eg: y/x.

3) a variable can be replaced by a function expression as long as the function expression does not contain the variable being matched.

eg: f(y)/x => Valid

f(x)/x => Invalid.

Example.

L1	L2	SUBST
knows (John, x)	knows (John, Jane)	Jane/x.
knows (John, x)	knows (y, OJ)	John/y, OJ/x.

L1	L2	SUBST	
knows (John, x)	knows (y, mother (y))	John/y	mother(John)/x
knows (John, x)	knows (x, OJ)	FAIL [x is replaced by 2 values ① John/x ② OJ/x } ⇒ invalid	
tryassassinate (Marcus, caesar)	hate (Marcus, caesar)	FAIL [Predicates are different]	
P(x, y, z)	P(x, y)	FAIL [no. of arguments are different]	

Eg: 2

For a given two literals to match, we can get many substitution sets.

hate (x, y)

hate (Marcus, z)

Substitution sets

1) Marcus/x, z/y } are equivalent

2) Marcus/x, y/z }

3) Marcus/x, caesar/y, caesar/z ⇒ more restrictive

The third although produces a match, also produce a substitution that is more restrictive than absolutely necessary for the match. Because, the final substitution produced by the unification process will be used by the resolution procedure, it is useful to generate the most general unifier possible.

Eg1: (Resolution for Predicate Logic).

1. Marcus was a man

Predicate Logic: man(marcus)

CNF: man(marcus)

2. Marcus was a pompeian

PL: Pompeian(marcus)

CNF: pompeian(marcus)

3. All pompeian were Romans

PL: $\forall x$: pompeian(x) $\rightarrow$ Roman(x)

CNF: $\neg$pompeian(x) $\vee$ Roman(x_1)

4. Caesar was a ruler

PL: ruler(caesar)

CNF: ruler(caesar)

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

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

CNF: $\neg$Roman(x_2) $\vee$ loyalto(x_2, caesar) $\vee$ hate(x_2, caesar)

6. Everyone is loyal to someone

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

CNF: loyalto(x_3, f(x_3))

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

PL: $\forall x$: $\forall y$: man(x) $\wedge$ ruler(y) $\wedge$ trytoassassinate(x, y) $\rightarrow$ $\neg$loyalto(x, y)

CNF: $\neg$man(x_4) $\vee$ $\neg$ruler(y_1) $\vee$ $\neg$trytoassassinate(x_4, y_1) $\vee$ $\neg$loyalto(x_4, y_1)

8. Marcus tried to assassinate caesar

PL: trytoassassinate(Marcus, caesar)

CNF: trytoassassinate(Marcus, caesar)

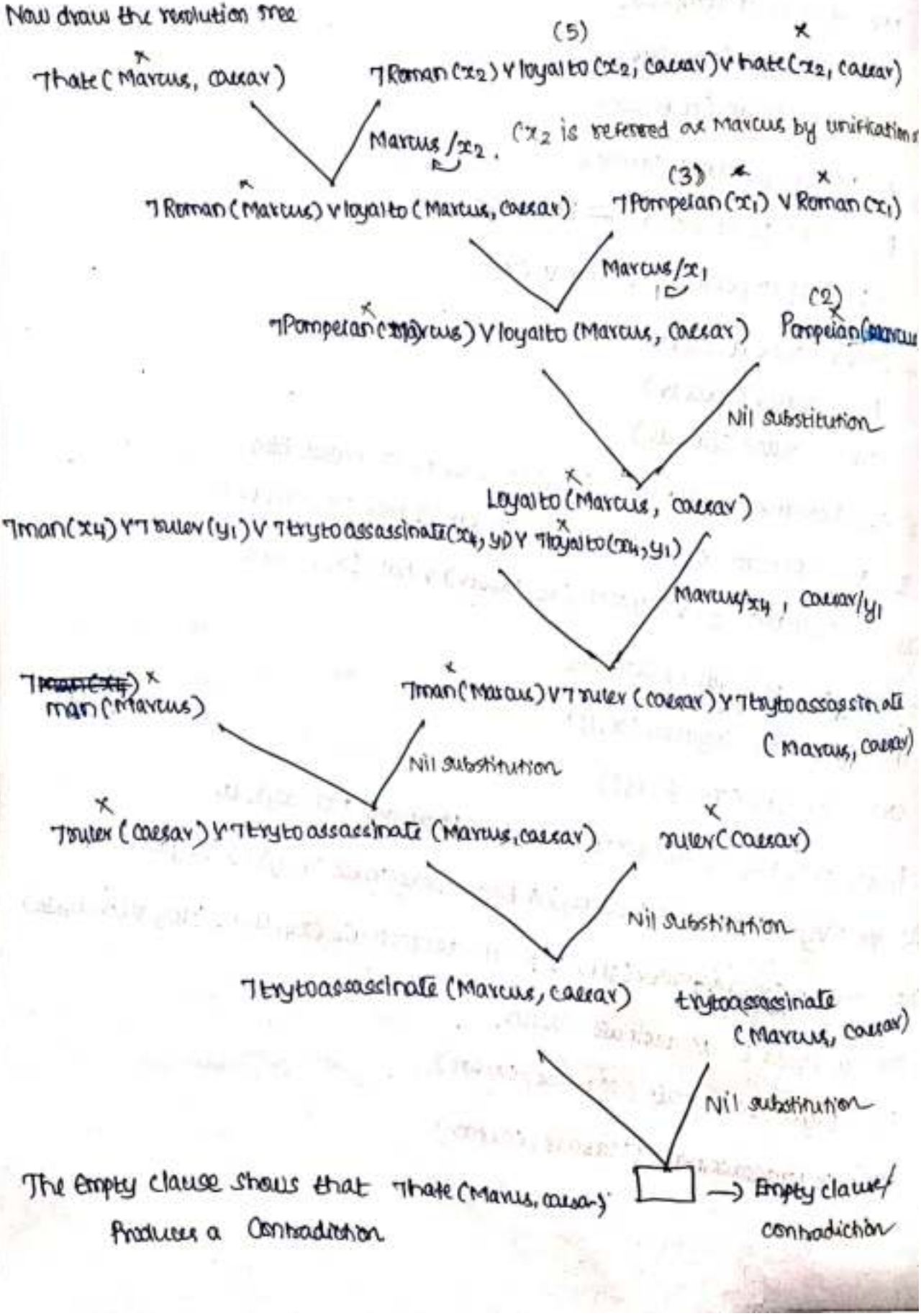
To Prove: Marcus hate caesar

PL: hate (Marcus, caesar)

CNF: hate (Marcus, caesar)

Negate the CNF → ¬hate (Marcus, caesar)

Now draw the resolution tree

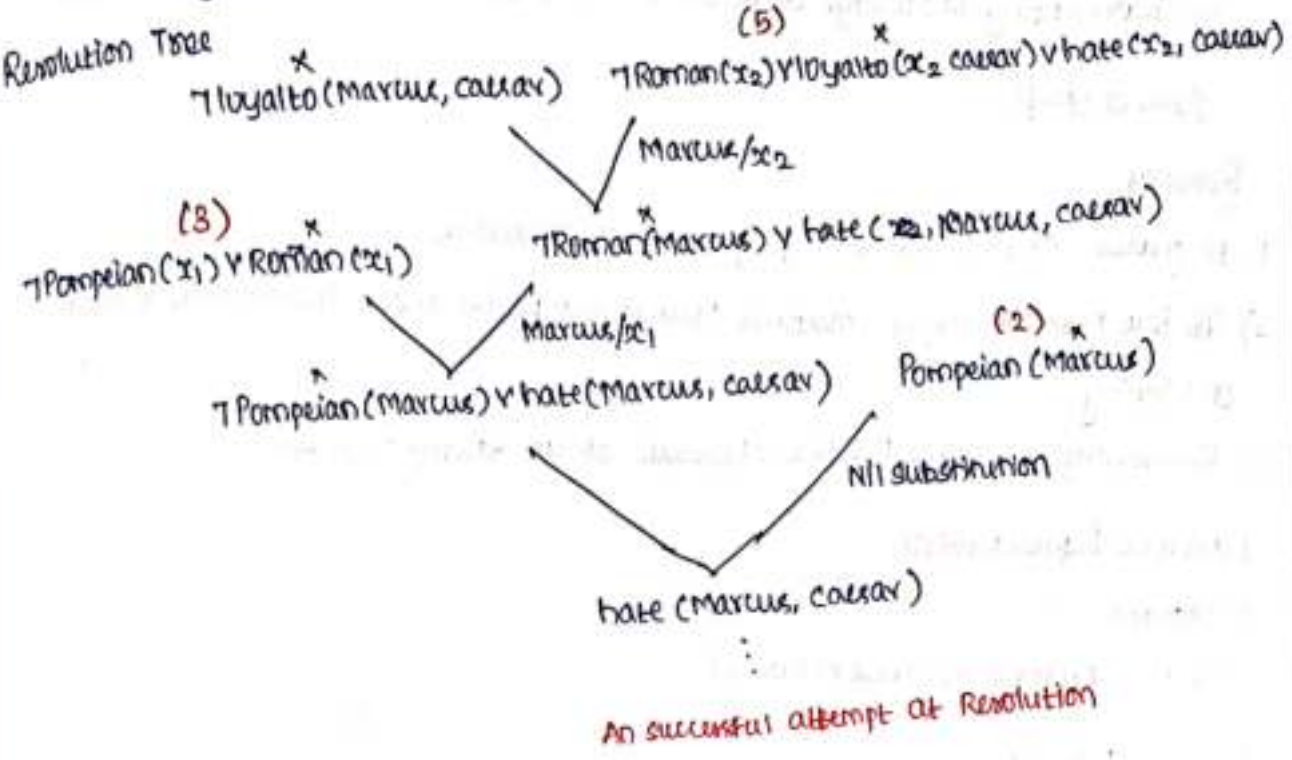


The Empty clause shows that ¬hate (Marcus, caesar) Produces a Contradiction

To Prove: loyalto(Marcus, caesar)

Negate the CNF → ¬loyalto(Marcus, caesar)

Resolution Tree



An successful attempt at Resolution

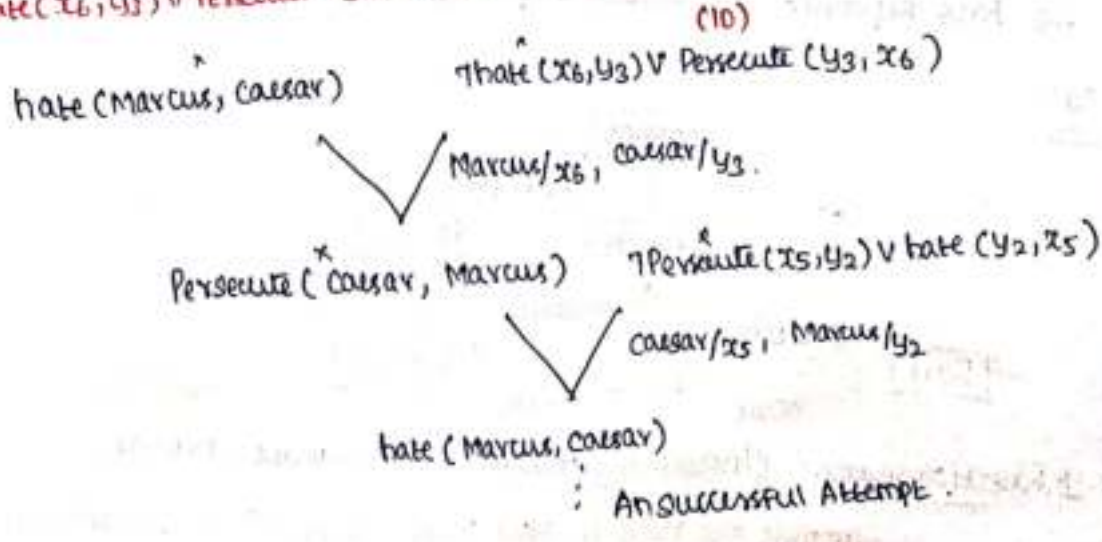
New Additional statements

9. Persecute(x,y) → hate(y,x) (Predicate logic)
10. hate(x,y) → Persecute(y,x) CNF form

↓ CNF

9. ¬Persecute(x5, y2) V hate(y2, x5)
10. ¬hate(x6, y3) V Persecute(y3, x6)

since x1, x2, x3, x4 are in statement (1 to 8).
∴ a → b ⇒ ¬a V b.



An successful Attempt.

Structured Representation of Knowledge

→ Representing knowledge as a set of entities and their attributes in the form of graph

Reasons:

1) It makes easy to describe properties of relations
2) It is a form of object-oriented Programming that shows modularity & ease of viewing
3) Retrieving the value for the attribute of an entity is fast.

Different Representation

1) Semantic Net
2) Frames (will be discussed in Unit 3)

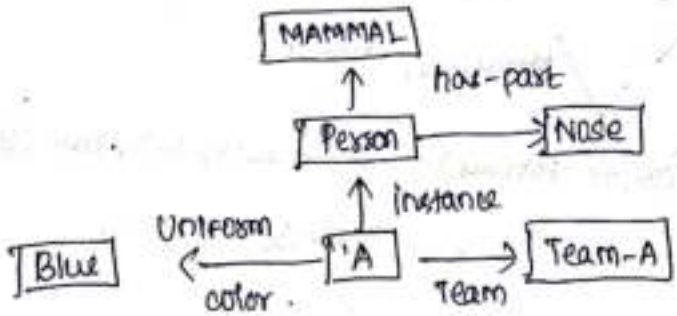
Semantic Network

Graphical knowledge Representation, It is basically developed to model human memory. It is use in neural network & expert system.

It consists of nodes and arcs.

→ Node Represent Information / object
→ Arcs Represent the relationships between Nodes.

eg:



Intersection search: (Inferencing mechanism of Semantic network)

Semantic net helps to find relationships among objects by spreading activation out from each of the two nodes and seeing where the activation met.

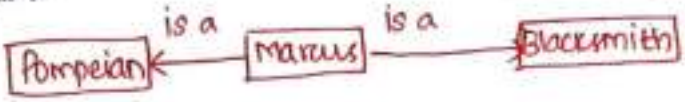
Examples

1) Construct semantic net representation

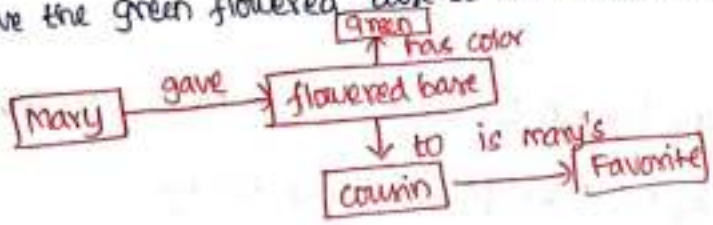
a) Pompeian (Marcus), Blacksmith (Marcus)

↑ Object

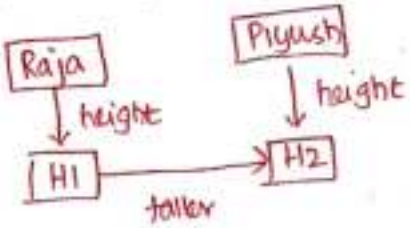
↓ class



b) Mary gave the green flowered base to her favourite cousin

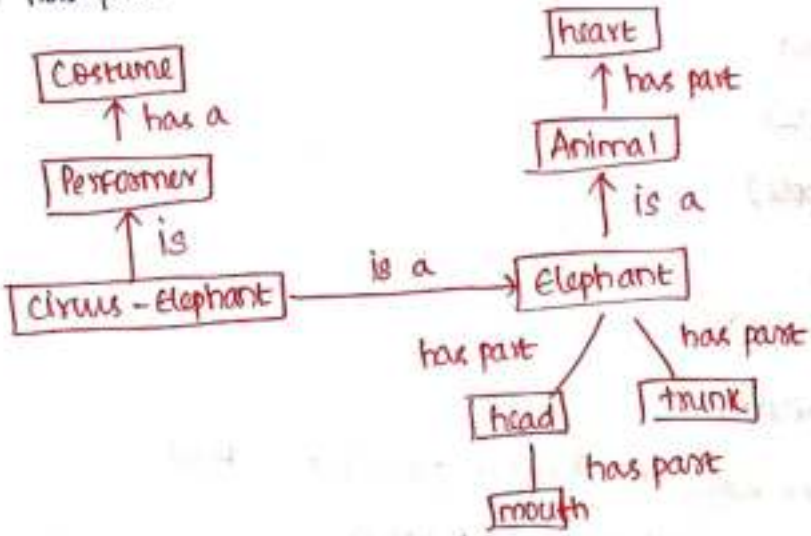


c) Raja is taller than piyush



2) Represent the following information in a semantic net

i) is a circus-Elephant elephant
ii) has part elephant head
iii) has part elephant trunk
iv) has part head mouth
v) is a Elephant animal
vi) has part animal heart
vii) is a circus-Elephant Performer
viii) has a performer costume



Advantages

1. Easy to translate to Prolog
2. Simple and easy to understand
3. Relationships are handled by pointers
4. Provides good visualization

Disadvantages

1. Represent Relationships between the objects which are only binary relationships
2. Nodes and Arcs may not have perfect values Always.

Reason for going to CNF (About clause form)

1. To make predicate logic into flatter i.e there was less embedding of components.
2. The quantifiers were separated from the set of the formula so that they did not need to be considered.

Example 2: Resolution in Predicate Logic.

i) John likes all kinds of Food.

P.L: $\forall x : Food(x) \rightarrow likes(John, x)$

CNF: $\neg Food(x) \vee likes(John, x)$

2) Apples are Food.

P.L: Food(apple)

CNF: Food(apple)

3) Chicken is Food

P.L: Food(Chicken)

CNF: Food(Chicken)

4) Anything Anyone eats and does not get killed is Food

P.L: $\forall x : \forall y : eats(x,y) \wedge alive(x) \rightarrow Food(y)$

CNF: $\neg eats(x_1,y) \vee \neg alive(x_2) \vee Food(y)$

5) Bill eats peanuts and is still alive

P.L: eats (Bill, Peanuts) ∧ alive (Bill)

CNF: eats (Bill, Peanuts) ∧ alive (Bill)

6) Susan eats anything Bill eats

P.L: ∀x : eats (Bill, x) → eats (Susan, x)

CNF: ¬eats (Bill, x_3) ∨ eats (Susan, x_3)

In Resolution number it x_1, x_2 . . .

Top The CNF Forms From the given statement are

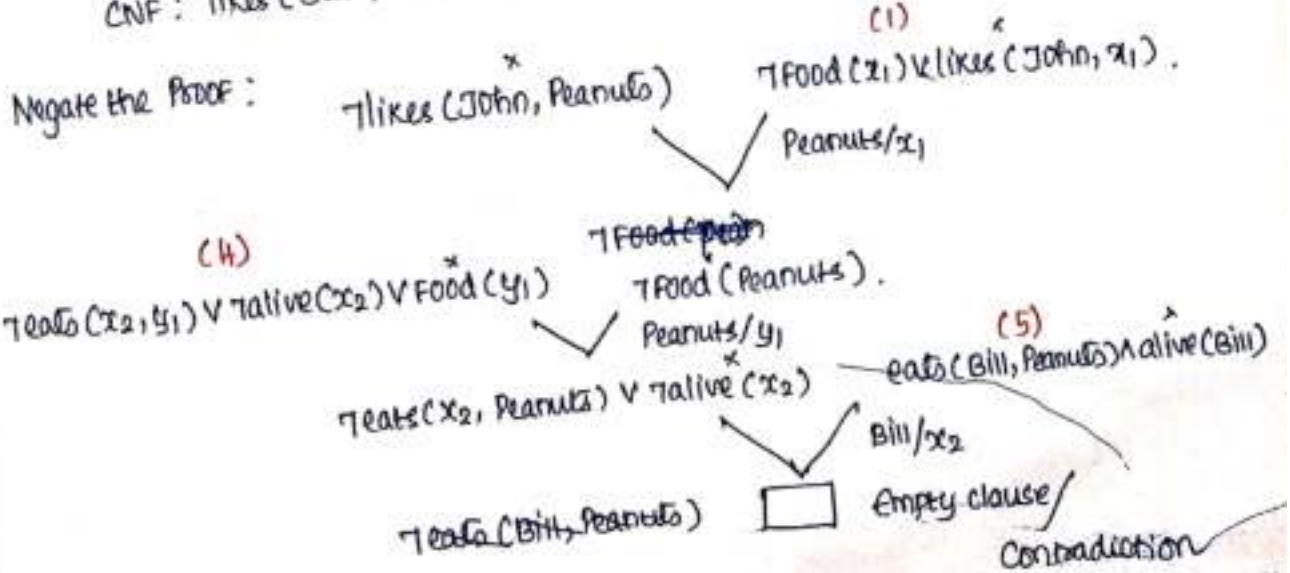
1. ¬Food (x_1) ∨ likes (John, x_1)
2. Food (apple)
3. Food (chicken)
4. ¬eats (x_2, y_1) ∨ ¬alive (x_2) ∨ Food (y_1)
5. eats (Bill, Peanuts) ∧ alive (Bill)
6. ¬eats (Bill, x_3) ∨ eats (Susan, x_3)

To Prove: John likes Peanuts

P.L: likes (John, Peanuts)

CNF: likes (John, Peanuts)

Negate the Proof: ¬likes (John, Peanuts)



The Empty clause shows that ¬likes (John, Peanuts) Produces a contradiction.

Unit-3

Knowledge Inference

Knowledge Representation:

Knowledge:

↳ Knowledge is an useful term to judge the understanding of an individual on a given subject.

↳ In Intelligent systems, domain is the main focused subject area. So the system specifically focuses on acquiring the domain knowledge.

Representation

A subarea of Artificial Intelligence concerned with understanding, designing and implementing ways of representing information in computer so that Programs (agents) can use this information.

→ To derive information that is implied by it.
→ To converse with people in natural language
→ To decide what to do next
→ To plan future activities
→ To solve problems in areas that normally require human expertise.

Knowledge - Based system / Knowledge Base
\ Inference Engine

Knowledge base

- A set of sentences that describe the world in some Formal (representational) language (eg. first-order logic)
- Domain specific knowledge

Inference Engine

- A set of procedures that work upon the representational language and can infer new facts or answer KB queries (eg. resolution algorithm, forward chaining)
- Domain independent

Prove : $\exists x$: hate (Marcus, x) $\wedge$ ruler(x)

(Negate): $\neg \exists x$: hate (Marcus, x) $\wedge$ ruler (x)

CNF : $\neg$hate (marcus, x) $\vee \neg$ ruler (x)

~~$\neg$hate (Marcus, x)~~ $\vee \neg$ ruler(x)

Production based system / Inferential system / Rule-based systems / Productions.

Definition:

Production system provides a AI Programs a structure to facilitate describing and performing search process.

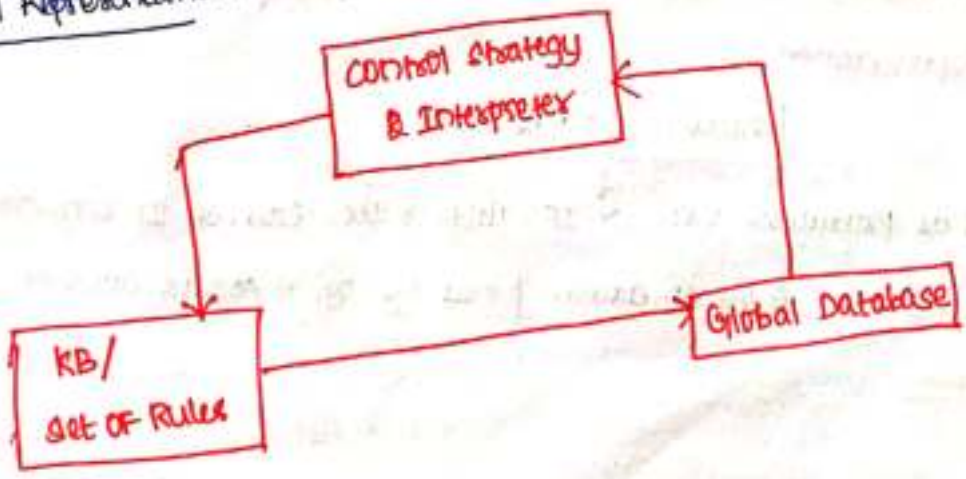
(or)

If a system adopts with "Production Rule" and a "ruler - interpreter" then the system is known as production system.

Production system consists of

1) A set of Rules / knowledge Base
2) A Database / knowledge [i.e system's short term memory / working memory]
3) Control strategy / control structure
4) A Rule Applier / Interpreter

Pictorial Representation



1) Set of Rules / knowledge Base:

Production rules are popular knowledge Representation structure. OPS5 (A Production system language).

OPS → Official Production System)

A Rule is an ordered pair of symbol strings [Conditional IF-THEN

The knowledge base is divided into

→ A Rule base (includes rules)

→ A working memory (includes facts)

Rules: A special type of IF-then rule

$P_1 \wedge P_2 \wedge P_3 \wedge \ldots \wedge P_n \Rightarrow a_1, a_2, a_3, \ldots, a_n$

LHS [Condition part / Precondition] that determines the applicability of the rules.

RHS [Action part] describes the operation to be performed if the rule is applied.

Syntax

IF < antecedent-1 >
< antecedent-2 >
:
THEN < decision / consequent >

Antecedent

A conjunction of conditions

Consequent

A sequence of Actions

Pictorial Representation

situation → action

→ called as production rules or IF_THEN rules. Each of IF condition is called as clause(s). A set of clause joined by logical AND is called as HORN clause.

Eg:

1. Rule for the University grading system

IF the marks obtained are greater than 75

THEN declare the student as in First class.

2. Global Database / Working memory / Short-term Memory

It is the central data structure used by the Production system. It contains information relevant for the given problem / Particular task. Here some parts of databases are permanent, other parts may be temporary and exist during the solution of the current problem.

→ This is a Dynamic structure.

3. Control Strategy / Control Structure

control strategy decides which rule to apply next during the process of searching for a solution to a problem.

(or)

It determines the order in which the rules will be compared to the database and provides a way of resolving any conflicts that can arise when several rules match at once.

4. Rule Applier / Interpreter

A computational system that implements the control strategy and applies the rule. It works based on Recognize-Act-cycle

This cycle has 4 stages.

1) Match : The condition in the rules against the elements in the working memory to identify the set of Applicable Rules.

2) Conflict Resolution: If there is more than one rule that can be applied, then use conflict Resolution, this strategy to choose which one to apply.

3) Apply : The chosen rule, which results in new state to the working memory.

4) Check : If the terminating condition is fulfilled, if it is / then it stops, otherwise return to stage 1.

(8) Note : The termination condition can either be defined by a goal state or by some kind of resource / time limitation.

Production System Algorithm

DATA (Binded with Initial Global Database)

When DATA satisfies the halting condition do

Begin

Select some Rule R that can be applied to DATA

return DATA

end.

Eg: sorting string Pattern.
Producting Rule for Water-Jug } Refer Unit-1.

Frame Based System / slot and Filler system

Organizing the knowledge in the form of packets is called as Frames. Frames are collection of slots (i.e attributes) which have certain values. Frame is a data structure that has slots for various objects and these slots contain some values.

The slots are described with attribute-value pairs <slot-name, value> and sometimes include instructions on how to apply or use the slot values.

The slots can be of any type and any size. The value of the slot can be

→ Primitive [text string, constants or Integer]
→ May be any other Frame or contain methods
→ Default values, conditions for filling a slot
→ subfields called facts (Facts may have names & values)
→ Pointers to other related frames

When frame is used.

[Natural language Understanding require inference i.e assumptions about what is typically true of the objects or situations under consideration, such information is coded into structures known as frames].

Frame:

A frame is a collection of attributes or slot and associated values that describe some real world entity.

Eg: Book

Slot	Filler
Publisher	McgrawHill
TITLE	AI
Author	Sanjay

Real world Entity

General syntax:

(<Frame Name>

(<slot 1> (Fact1 <value1> . . . . <value K>)

(Fact2 <value1> . . . . <value K>)

(Fact 3 <value1> . . . . <value K>))

(<slot 2> (Fact1 <value1> . . . . <value K>)

Example of frame of a university Department is

)

(DEPARTMENT → Frame name.

(FACULTY → Slot name (Number (VALUE 8)) → Facets / Facts

(Education (VALUE Ph.D)))

(CLASSROOM (Number (VALUE 10))

(CAPACITY (VALUE 60)))

(UNIVERSITY (VALUE AU))

)

When discussing about particular Person

(abc)

(Profession (VALUE Engineer))

(Age (VALUE 25))

(City (VALUE Chennai))

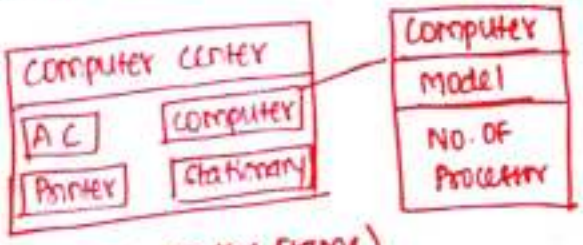
(State (VALUE TN))

Types of Frames:

1. The Frame above consists only descriptive type of knowledge, therefore called as declarative frames

[A Frame that just contains only description about objects

Eg: Frame for Computer center



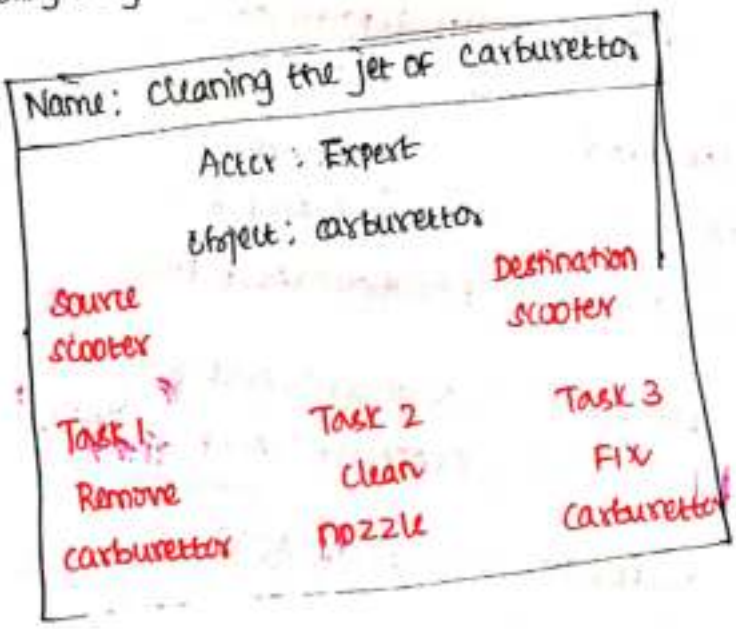
(only the name of the object of the frame)

(i.e) what this object is going to do, functions of object.

2. Procedural Frames:

A frame may contain knowledge about actions or Procedure, then this frame is called as Procedural frame

Eg: cleaning the jet of carburettor.



Reasoning with Frames

A Frame can be attached with another frame and can create a network of Frames. [which looks like semantic network / Associative N/w's. This network helps to easily implement Property inheritance / Default Reasoning.

Q1

(ford → A kind of
(AKO (Value car))

(GAS_MILEAGE (DEFAULT fget)) → is a function call to fetch a default value from another frame.

(RANGE (VALUE if_needed))

(IS A (VALUE FORD)) → Procedure/demons
→ inheritance value.

Procedures are

1) IF_NEEDED: is invoked when it is necessary to acquire a slot value.
2) IF_CHANGED: is invoked when a new value of a slot is changed.
3) IF_ADDED: is invoked when a value is added to a slot
4) IF_REMOVED: is invoked when a value of a slot is deleted.

Eg: For IF_ADDED Demon

(Avail_ticket
(Traveller (value if_added))
)

↓

if_added
? if traveller : age < 86 then
discount = 50%.
else discount = 0%.
;.

Q.

Generic Frame for property

slots	Filler
name	Property
specialization_OF	a_kind_OF object
types	(car, boat, house)
	if_added: Procedure ADD_PROPERTY
owner	default: government
	if_needed: Procedure FIND_OWNER
location	(home, work, mobile)

	status	(missing, poor, good)
	under-warranty	(yes, no)

Car Frame – A Generic subframe of property

	slots	Fillers
c	name	car
	specialization-of	a-kind-of property
	types	(sedan, sports, convertible)
Slot fillers may also contain relations. Ex: a-kind-of & is-a	manufacturer	(GM, Ford, Toyota)
	location	mobile
	wheels	4
	transmission	(manual, automatic)
	engine	(gasoline, hybrid gas/electric)

Pictorial Representation of N/w of Frames

Refer inheritable knowledge in unit-II

(Baseball player representation)

Frame-Based Representation language

Host language is LISP (List Processing), developed due to popularity of Frame Representation. They have functions to create, access, modify, update and display frames.

Function to create a frame

(Fdefine F-name <parents> <slots>)

where

Fdefine:- frame definition function

Fname:- Name of the frame

<parents>: list of parents

<slots>: list of names of slots and the initial values.

Frame name → Parent name / base class

Eg: (Define General-train land-transport
(type (value passenger))
)

→ Few Functions, Provided in Frame language.

1. (fget Frame slotname facetname) – returns data from specified location.
2. (fslots Frame) – returns name of slots
3. (ffacets Frame slotname) – returns names of facets
4. (fput Frame slotname facetname) – adds data to a specified location
5. (fremove frame slotname facetname) – Removes data from specified location.

```
(Elephant
   <: is-A Mammal>
   <: color gray>. . )
(Royal Elephant
   <: is-A Mammal>
   <: color white>)
(& clyde
   <: Instance-OF Royal elephant>)
```

(Object is an instance OF class)

Inference (Forward chaining and Backward chaining)

Inference: (conclusion/decision)

It is a component of the system that applies logical rules to the knowledge base (KB) to reduce new information.

It works on two modes.

1. Forward chaining/Reasoning
2. Backward chaining/Reasoning

There are two directions available to discover a path through a problem space, from initial state to a goal state

→ Forward chaining => From start state

→ Backward chaining => From goal state

1. Reasoning Forward From Initial state

→ Begin building a tree by starting with initial state at the root of the tree.

→ The next level of the tree is generated by finding the rules where left-side matches the root node and their right-side to create a new node.

→ Repeat the above step until goal state is reached.

Definition:

The left-side [i.e the precondition] are matched against the current state and the right side [i.e action] are used to generate new nodes until the goal is reached.

2. Reasoning backward From goal state / goal-directed Reasoning

→ Begin building a tree by ~~matching~~ making goal state as root node of the tree

→ Generate the next level of tree by finding all the rules whose right-side matches the root node and their left-side to create a new node.

→ Repeat / continue until a node that matches the ~~tree~~ initial state is generated.

Definition

The right-side are matched against the current state and the left sides are used to generate new node until the initial state is reached.

Forward chaining Algorithm

function FOL_FC_ASK (KB, α) returns a substitution or false

inputs: KB, the knowledge base
α, the query an atomic sentence — Is a type of declarative sentence which is either true or false & which cannot be broken down into other simpler sentences.

local variable: new, the new sentence infered on each iterator

Repeat until new is empty

new ← { }

for each rule in KB do

{ $(P_1 \wedge P_2 \ldots \wedge P_n) \rightarrow q$) ← STANDARDIZE_VARIABLES (rule)

for each θ such that SUBST (θ, $P_1 \wedge P_2 \wedge \ldots \wedge P_n$) = SUBST (θ, $P_1' \wedge \ldots P_n'$) for some $P_1' \ldots P_n'$ in KB

$q' \leftarrow$ SUBST (θ, q)

IF q' does not unify with some sentence already in KB or new then

add q' to new

$\phi \leftarrow$ UNIFY (q', α)

if φ is not fail then return φ

add new to KB

return false

o/p from unification algorithm is θ.

q → Implies the Action

new → new infered statement.

Example:

Consider the following sentence

1. John likes all kinds of food
2. Apples are food
3. Chicken is food
4. Anything Any one eats and isn't killed by this food
5. Bill eats peanuts and is still alive
6. Sue eats everything Bill eats

Prove that John likes peanuts using Forward chaining.

Step1:

Convert the given statement / facts to its equivalent FOL

1. John likes all kinds of food.

   $\forall x$: Food(x) $\rightarrow$ likes (John, x)

2. Apples are food [Variables is don't take only constants].

   Food (Apples) so, 2, 3, 5

3. Chicken is Food

   Food (chicken)

4. Anything Anyone eats and isn't killed by is food

   $\forall x$: ($\exists y$: eats (y, x) $\wedge$ $\neg$killed by (y, x)) $\rightarrow$ Food (x)

5. Bill eats peanuts and is still alive

   eats (Bill, peanuts) $\wedge$ $\neg$killed by (Bill, Peanuts)

6. Sue eats everything Bill eats

   $\forall x$: eats (Bill, x) $\rightarrow$ eats (Sue, x)

Step2: Standardize variables [Already in algorithm a statement is added to standardize values].

~~Step~~ :

Step3: Algorithm starts to prove John likes peanuts using Foward chaining.

KB $\rightarrow$

1. $\forall x$: Food (x) $\rightarrow$ likes (John, x)
2. Food (Apples)
3. Food (chicken)
4. $\forall x$: ($\exists y$: eats (y, x) $\wedge$ $\neg$killed by (y, x)) $\rightarrow$ Food (x)
5. eats (Bill, peanuts) $\wedge$ $\neg$killed by (Bill, peanuts)
6. $\forall x$: eats (Bill, x) $\rightarrow$ eats (Sue, x)

α → query in predicate logic.

John likes peanuts

α → likes (John, peanuts) → Predicate logic

(variable)

Initially new is empty

for each rule in KB (knowledge Base)

$P_1 \wedge P_2 \wedge \ldots \wedge P_n \rightarrow q$

As it is forward chaining we need to start from the initial state.

α = likes (John, Peanuts)

Initial node

eats (Bill, peanuts) $\wedge$ $\neg$killed by (Bill, Peanuts) [start with statement which have w/o implication]

P_1' P_2'.

Now, we need to compare $P_1 \wedge P_2 \rightarrow q$ (statement) in the rule, since the left side matches with the root node and the right side is the new node (generated)

According to the following statement in algorithm

For each θ such that SUBST(θ, $P_1 \ldots P_n$) = SUBST(θ, $P_1' \ldots P_n'$)

For some $P_1' \ldots P_n'$ in KB.

We need to check to the statement in KB (the left-side should be equal to the initial root) in KB Statement / Rule 4 matches with the initial node

P_1 P_2 q

Rule 4 => eats (y_1, x) $\wedge$ $\neg$killed by (y_1, x) → Food (x)

Initial node => eats (Bill, Peanuts) $\wedge$ $\neg$killed by (Bill, Peanuts)

Standardize variables in Rule 4

=> eats (y_1', x_1') $\wedge$ $\neg$killed by (y_1, x_1)

By substitution Bill/y_1 and Peanuts/x_1

θ → Bill/y_1 and Peanuts/x_1

O/p from unification algorithm

Now, $q' \leftarrow$ SUBST(θ, q)

(From Rule 4) q → Food (x_1)

q' → Food (peanuts). (after substitution)

Now add q' to new

new ← Food (peanuts)

ϕ ← UNIFY (q', α)

q' ← Food (peanuts)

α ← likes (John, peanuts)

} It does not return ϕ since. Predicates are different.

Now add q' to the KB,

so the 7th statement is

7. Food (peanuts)

If q' is not null, then we need to move the next step.

Step3:

Now, the knowledge base has 7 statements. (By adding Food(peanuts))

Again it checks for the predicate Food.

It is available in Rule 1 => Food(x) ⟶ likes (John, x)

↓ standardize variables

Food(x_2) ⟶ likes (John, x_2)

P q.

By substitution Food(x_2)

Food (peanuts) => peanuts/x_2.

θ <= peanuts/x_2

(From Rule 1) => q ← likes (John, x_2)

↓

q' ← likes (John, peanuts) (after substitution)

add q' to new

new ← likes (John, peanuts)

ϕ ← UNIFY(q', α)

q' ← likes (John), peanuts)

α ← likes (John, peanuts)

} Since the predicate & argument are same it returns ϕ.

It is the goal state, it returns ϕ.

Tree Representation

Add on Facts(new)

Food (peanuts)

likes (John, Peanuts)

P_1' P_2'

eats (Bill, peanuts) ∧ ¬killedby (Bill, Peanuts)

↓ Peanuts/x_1, Bill/y_1

Rule 4 ⇒ Food (peanuts)

↓ Peanuts/x_2

Rule 1 ⇒ likes (John, peanuts)

↓

goal state.

Pictorial view: (Forward chaining)

~~eats (John, Apples)~~ → Not proved

likes (John, Apples)

Not proved.

likes (John, Chicken)

(same Predicate)

∀x: ∃y: eats (y, x) ∧ ¬killedby (y, x) → Food (x)

Apples/x_1

Food (apples) | eats (Bill, Peanuts) | ¬killedby (Bill, Peanuts) | Food (chicken).

likes (John, peanuts) [Rule 1]

Peanuts/x_2.

Food (Peanuts) [Rule 4].

Bill/y_1; Peanuts/x_1

eats (Bill, Peanuts) | ¬killedby (Bill, Peanuts)

Bottom to Top

Algorithm For Backward chaining

Function FOL_BC_ASK (KB, goals, θ) returns a set of substitution

Inputs: KB, a knowledge base

(Statement to be Proved) → goals, a list of conjuncts forming a query (θ already applied)

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

Local Variables: answers, a set of substitution, initially empty

if goal is empty then return θ

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

For each sentence r in KB

where STANDARDIZE_APART(r) = $((P_1 \wedge \ldots \wedge P_n) \rightarrow q)$

and $\theta' \leftarrow$ UNIFY(q, q') succeeds

new goals $\leftarrow$ $[P_1 \ldots P_n \,||\, \text{REST(goals)}]$

answers $\leftarrow$ FOL_BC_ASK(KB, newgoals, compose(θ', θ) ∪ answers

return answers.

Eg The statements are considered as

1. John likes all kinds of food
2. Apples are food
3. Chicken is food
4. Anything Anyone eats and isn't killed by his food.
5. Bill eats peanuts and is still alive
6. Sue eats everything Bill eats

In Predicate logic

1. $\forall x : Food(x) \rightarrow likes(John, x)$
2. Food(apples)
3. Food(chicken)
4. $\forall x \forall y : eats(y,x) \wedge \neg killedby(y,x)) \rightarrow Food(x)$
5. eats(Bill, peanuts) $\wedge \neg$ killedby(Bill, peanuts)
6. $\forall x : eats(Bill, x) \rightarrow eats(Sue, x)$

Algorithm:

It checks for $P \rightarrow q$; q. Right (side) → matched

left (side) → new node.

goal => likes(John, Peanuts)

BC(KB, goals, θ)

↓ Initially

BC(KB, likes(John, Peanuts), $\frac{\{\}}{\theta}$ y.

First pass.

$\theta = \{\}$, goal = likes(John, Peanuts), First(goals) = likes(John, Peanuts)

Here goal is not empty, θ=

$q' \leftarrow SUBST(\theta, First(goals))$

$\leftarrow SUBST(\{\}, likes(John, Peanuts))$.

$q' \leftarrow likes(John, Peanuts)$

Standardize the variables in Predicate logic ($(P_1 \wedge P_2 \ldots \wedge P_n) \rightarrow q$)

In Rule => $Food(x) \rightarrow likes(John, x)$

↓ standardize

$Food(x_1) \rightarrow likes(John, x_1)$

$P_1 \wedge P_2 \ldots$ q

left side (new node) Right side (matches with q')

$q \leftarrow likes(John, x_1)$

Now $\theta' \leftarrow UNIFY(q, q')$ => UNIFY(likes(John, x_1), likes(John, Peanuts

$\theta' \leftarrow Peanuts/x_1$

newgoals ← [$P_1 \ldots P_n$ || REST(goals)]

Rest(goals) → null
Now no other goals.

Here $P_1 \ldots P_n$ => Food(x_1)

newgoals ← Food(x_1) || NULL (concatination)

newgoals ← Food(x_1)

answers ← FOL_BC_ASK(KB, newgoals, compose(θ, θ)

FOL_BC_ASK(KB, Food(x_1), compose(Peanuts/x_1, NULL)

FOL_BC_ASK(KB, Food(x_1), Peanuts/x_1)

Again it checks, until goal is empty.

Second pass

goals = Food(x_1), θ = Peanuts/x_1 ; First(goal) = Food(x_1)

Here goal is not empty

q' ← SUBST(θ, First(goal))

← SUBST(Peanuts/x_1, Food(x_1))

After substituting x_1 ← Peanuts

q' ← Food(peanuts)

Standardize the variables in predicate logic ($P_1 \wedge P_2 \wedge \ldots \wedge P_n \rightarrow q$)

check for the value Food(peanuts) in the Rule (KB) where Right side should match.

From KB Rule (RB)4 Match. => eats(y_1, x_2) ∧ ¬killedby(y_1, x_2) → Food(x_2)

$P_1 \wedge P_2$ → q

left side Right side

q ← Food(x_2)

Now θ' ← UNIFY(q, q')

← UNIFY(Food(x_2), Food(peanuts))

The unification algorithm will work since predicate matches.

θ' ← Peanuts/x_2

newgoals ← [P1 ... Pn || Rest(goals)]

Here P1 ... Pn = eats(y1, x2) ∧ Killedby(y1, x2)

newgoals ← eats(y1, x2) ∧ Killedby(y1, x2) || NULL.
(since Rest(goal) is null.)

newgoals = eats(y1, x2) ∧ Killedby(y1, x2)

answers = FOL-BC-ASK(KB, newgoals, compose(θ', θ))

= FOL-BC-ASK(KB, eats(y1, x2) ∧ Killedby(y1, x2), compose(Peanuts/x2, Peanuts/x1))

= FOL-BC-ASK(KB, eats(y1, x2) ∧ Killedby(y1, x2), {Peanuts/x2, Peanuts/x1})

First Pass

goals = eats(y1, x2) ∧ Killedby(y1, x2) ; θ = Peanuts/x2, Peanuts/x1 ;

First(goal) = eats(y1, x2) ∧ Killedby(y1, x2)

Rest goal is not empty

q ← SUBST(θ, First(goal))

← SUBST(Peanuts/x2, Peanuts/x1, eats(y1, x2) ∧ Killedby(y1, x2))

After substituting x2 ← peanuts

q' = eats(y1, Peanuts) ∧ Killedby(y1, Peanuts)

Standardize the variables in predicate logic (P1 ∧ P2 ... ∧ Pn) → q

Here the match should be made at Rightside(q) only one Rule

matches in the KB (i.e Rule 5) Consider the Rule as q.

5 ⇒ eats(Bill, Peanuts) ∧ Killedby(Bill, Peanuts)

Rightside q

q ← Eats(Bill, peanuts) ∧ Killedby(Bill, peanuts)

Now θ' ← UNIFY(q, q')

= UNIFY(eats(Bill, Peanuts) ∧ Killedby(Bill, Peanuts)
∧ eats(y1', Peanuts) ∧ Killedby(y1', Peanuts))

By unification,

$\theta' \leftarrow Bill/y_1$

newgoals $\leftarrow$ [$P_1 \ldots P_n$ || Rest(goals)]

Here $P_1 \ldots P_n$ = NULL [since no left side value from Rule 5]

and Rest(goals) = NULL

newgoals $\leftarrow$ [NULL || NULL]

newgoals = NULL.

answers $\leftarrow$ FOL_BC_ASK(KB, newgoals, compose(θ, θ'))

FOL_BC_ASK (KB, NULL, compose (Peanuts/x_2, Peanuts/x_1, Bill/y_1))

answers $\leftarrow$ FOL_BC_ASK (KB, NULL, {Peanuts/x_2, Peanuts/x_1, Bill/y_1})

Fourth Pass.

goals = NULL ; θ = {Peanuts/x_2, Peanuts/x_1, Bill/y_1}

First(goal) = NULL

Here goal is Empty IF goal is Empty ; return θ.

θ = { Peanuts/x_2, Peanuts/x_1, Bill/y_1 } .

Tree Representation:

Likes (John, peanuts) (Goal State)

$\downarrow$ Peanuts/x_1

Food(x_1) peanuts

$\downarrow$ Peanuts/x_2 Peanuts/x_1

eats(y_1, Peanuts/x_2) $\wedge$ KilledBy(y_1, Peanuts/x_2)

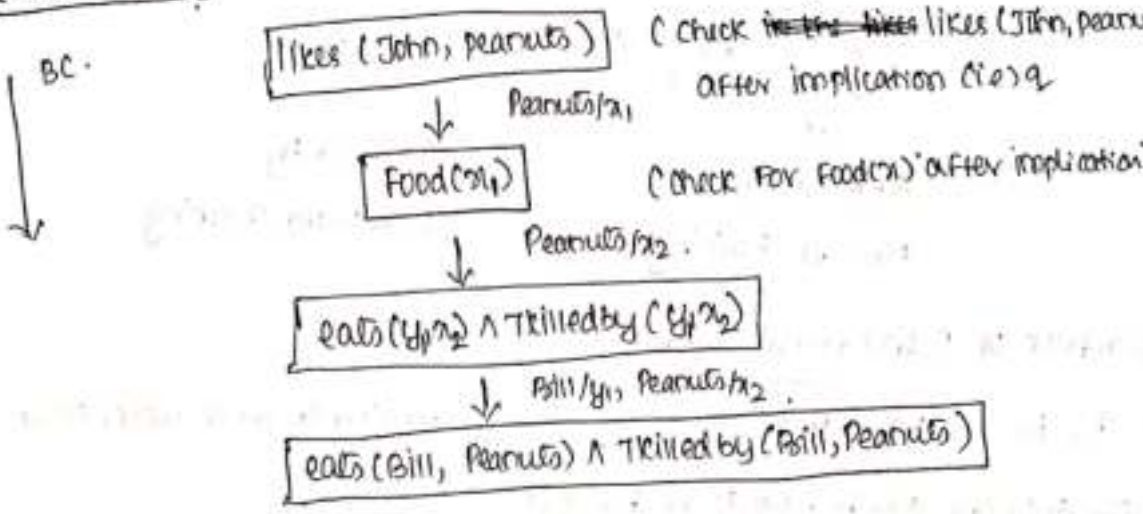
$\downarrow$ Bill/y_1 ; Peanuts/x_2 ; Peanuts/x_1

eats (Bill, peanuts) $\wedge$ KilledBy (Bill, Peanuts)

$\downarrow$ Peanuts/x_2 ; Peanuts/x_1, Bill/y_1.

NULL (Initial State)

Backward chaining



Reasoning with Uncertainty:

Reasoning: - Deriving conclusions from the available set of data & information

Eg: ~~King Dashavatha~~ Gokul was father of ~~Rama~~ Ram.

~~Rama~~ Ram was father of ~~two sons~~ Arun and Ajay

Conclusion: Gokul was grand father of Arun and Ajay.

Four Factors that helps to choose Forward chaining / Backward chaining

1) Are there more possible start state or goal state

Opinion: Like to start from smaller set of states to the larger set of rules.

2) In which direction is the branching factor greater? (the average number of nodes that are generated by a node)

Opion: Like to proceed in the direction with the lower branching factor.

3) Will the program be asked to justify its reasoning process to a user?

Opinion: proceed in the direction that corresponds more closely with the way the user will think.

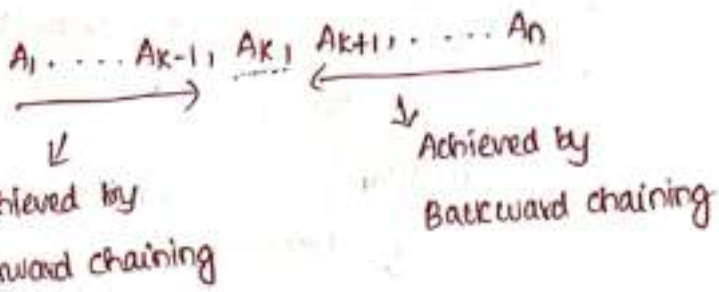
4) What kind of event is going to trigger a problem

opinion 1: If the problem is "arrival to new fact" → Apply Forward chaining

2: If the problem is "Answer for a Query" → Apply Backward chaining

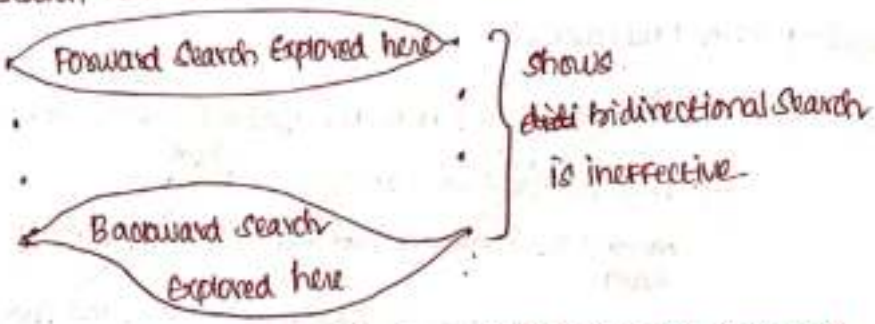
=> Searching (i.e Reasoning) can be done both using Forward and backward chaining (i.e simultaneous) from start state and goal state until two paths meet somewhere in between. This is called as Bidirectional Search.

Representation :



Drawback of Bidirectional Search

1) The two searches may cross each other, resulting in more work, that makes only one search have to be Finished.



2) If the problem is solved so separately by Forward chaining and backward chaining, the results can be encouraging

3) Same set of Rules can be used for both Forward and backward mapping / Reasoning

→ Forward Rules => which uses knowledge how to respond to certain input states.

→ Backward Rules => which uses knowledge how to achieve particular goal.

Reasoning with Uncertainty

Reasoning: Deriving conclusions from the available set of data and information.

Eg: King Raju was Father of Ram
Ram was Father of Geetha
Conclusion: King Raju was grand father of Geetha

8. IF (temperature is hot OR too-hot) and (target is warm) then command is cool.

9. IF (temperature is warm) and (target is warm) then command is no-change.

Advantages

1. Uses imprecise language.
2. Inherently Robust

Disadvantages

1. No systematic approach to fuzzy system design
2. Understandable only when simple.

The process of generating membership values for a fuzzy variable using membership functions → Fuzzification.

The process of transforming a fuzzy output of a fuzzy inference system into a crisp output → Defuzzification

---

1) American(x_1) ∧ weapon(y_1) ∧ sells(x_1, y_1, z_1) ∧ Hostile(z_1) → Criminal(x_1)

2) Owns(Nono, M

What is Backward chaining and how does it work? Explain the Backward chaining Algorithm for the following facts and prove that "West is a criminal".

1) It is a crime for an American to sell weapons to hostile nation.

American(x) ∧ weapon(y) ∧ sells(x, y, z) ∧ Hostile(z) → Criminal(x)

2) Nono owns some missiles

∃x owns(Nono, x) ∧ Missile(x) → Removing Existential Replace x by constant

owns(Nono, M1) ∧ Missile(M1)

3) All of its missiles were sold it to it by Colonel West

∀x: Missile(x) ∧ owns(Nono, x) → sells(West, x, Nono)

4) Missiles are weapons.

Missile(x) → weapon(x)

5) An enemy of America counts as "Hostile"

Enemy(x, America) → Hostile(x)

6) West, who is American

American (West)

7) The country Nono, an enemy of America

Enemy (Nono, America).

FOL

1. American (x) ∧ weapon(y) ∧ sells (x, y, z) ∧ Hostile (z) → Criminal(x)
2. Owns (Nono, M1)
3. Missile (M1)
4. Missile (x) ∧ Owns (Nono, x) → sells (West, x, Nono)
5. Missile (x) → Weapon(x)
6. Enemy (x, America) → Hostile(x)
7. American (West)
8. Enemy (Nono, America).

By Tracing with Algorithm.

It Checks for P → q, q

where the rightside → match

leftside → newnode.

Initially

BC (KB, goals, θ).

To Prove: West is a criminal

FOL: Criminal (West)

goal = Criminal (West)

θ initially Empty.

BC(KB, Criminal (West), {})

First pass

θ = {}, goals = Criminal(West), Firstgoals = Criminal (West)

Here goal is not empty

q' ← SUBST ({}, Criminal (West))

q' ← Criminal (West)

For each sentence s in KB

STANDARDIZE-APART(r) = $(P_1 \wedge P_2 \wedge \ldots \wedge P_n \rightarrow q)$

Check the Right side match for criminal(west)

In KB Rule => $\underbrace{American(x_1) \wedge weapon(y_1) \wedge sells(x_1, y_1, z_1) \wedge Hostile(z_1)}_{P} \rightarrow \underbrace{Criminal(x_1)}_{q}$

left-side (raw node)

Right-side matches with q'

$q \leftarrow criminal(x_1)$

NOW, $\theta' \leftarrow$ UNIFY(q, q')

$\leftarrow$ UNIFY(criminal(x_1), criminal(west))

$\theta' \leftarrow west/x_1$

REST goals → null.

newgoals $\leftarrow$ [$P_1 \ldots P_n$ || REST(goals)]

Here $P_1 \ldots P_n$ = $American(x_1) \wedge weapon(y_1) \wedge sells(x_1, y_1, z_1) \wedge Hostile(z_1)$

newgoals $\leftarrow$ $American(x_1) \wedge weapon(y_1) \wedge sells(x_1, y_1, z_1) \wedge Hostile(z_1)$ || NULL

newgoals $\leftarrow$ $American(x_1) \wedge weapon(y_1) \wedge sells(x_1, y_1, z_1) \wedge Hostile(z_1)$

answers $\leftarrow$ FOL_BC_ASK(KB, newgoals, compose(θ', θ))

FOL_BC_ASK(KB, $American(x_1) \wedge weapon(y_1) \wedge sells(x_1, y_1, z_1) \wedge Hostile(z_1)$,

Compose(west/x_1, NULL)

BC_ASK.(KB, $American(x_1) \wedge weapon(y_1) \wedge sells(x_1, y_1, z_1) \wedge Hostile(z_1)$,

west/x_1)

Again it checks until goal is empty

Second Pass

goals = $American(x_1) \wedge weapon(y_1) \wedge sells(x_1, y_1, z_1) \wedge Hostile(z_1)$; θ = west/x_1

Again goal is not empty

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

Check whether the Fact (whole fact) is available in the rule.

So, divide the facts

First(goals) = American(x_1), since θ = west/x_1

~~American(west)~~

$q' \leftarrow$ SUBST(west/x_1, American(x_1))

$q' \leftarrow$ American(west)

~~New θ' ← UNIFY(q, q')~~

← ~~UNIFY~~(

Check for the Rule (P → q, q)

Now, in Rule ⑦ from KB q ← American(west)

Now θ' ← UNIFY(q, q')

UNIFY(American(west), American(west))

θ' ← {}

newgoals ← [P1 ... Pn || REST(goals)]

P1 ... Pn = NULL

← REST(goals) ← Weapon(y1) ∧ Sells(x1, y1, z1) ∧ Hostile(z1)

newgoals ← [Weapon(y1) ∧ Sells(x1, y1, z1) ∧ Hostile(z1)]

answers ← FOL_BC_ASK(KB, newgoals, compose(θ', θ))

FOL_BC_ASK(KB, Weapon(y1) ∧ Sells(x1, y1, z1) ∧ Hostile(z1), compose(θ3, west/x1)}.

FOL_BC_ASK(KB, Weapon(y1) ∧ Sells(x1, y1, z1) ∧ Hostile(z1), west/x1}.

Again it checks for goal is Empty.

Third Pass

goals = Weapon(y1) ∧ Sells(x1, y1, z1) ∧ Hostile(z1), θ = {west/x1}

Again goal is not Empty

q' ← SUBST(θ, First(goals))

Since the whole fact is not available in the knowledge base, split the goal.

Firstgoals ← Weapon(y1)

q' ← SUBST(west/x1, Weapon(y1))

q' ← Weapon(y1).

Check in KB for the match, it is found in KB, Rule 5 ⇒ Missile(z2) → Weapon(z2)

P q

q ← Weapon(z2)

$\theta' \leftarrow$ UNIFY(q, q')

UNIFY(weapon(x_2), weapon(y_1))

L1 L2 L2/L1

After unification, the predicates are same,

$\theta' \leftarrow y_1/x_2$.

newgoals $\leftarrow$ [P1 ... Pn || REST(goals)]

REST goals $\leftarrow$ sells$(x_1, y_1, z_1) \wedge$ Hostile(z_1)

$P \leftarrow$ Missile(x_2)

newgoals $\leftarrow$ [Missile$(x_2) \wedge$ sells$(x_1, y_1, z_1) \wedge$ Hostile(z_1)].

answers $\leftarrow$ FOL_BC_ASK(KB, newgoals, compose(θ', θ)).

FOL-BC-ASK(KB, Missile$(x_2) \wedge$ sells$(x_1, y_1, z_1) \wedge$ Hostile(z_1), Compose$(y_1/x_2$, west$/x_1$)

BC_ASK (KB, Missile$(x_2) \wedge$ sells$(x_1, y_1, z_1) \wedge$ Hostile(z_1), $\{y_1/x_2$, west$/x_1\}$)

Again it checks, goal is not Empty

Fourth pass.

goals = Missile$(x_2) \wedge$ sells$(x_1, y_1, z_1) \wedge$ Hostile(z_1),

$\theta = \{y_1/x_2$, west$/x_1\}$

Again goal is not Empty.

$q' \leftarrow$ SUBST$(\theta$, First(goals))

since the whole fact is not available in the KB, split the goal.

First(goals) $\leftarrow$ Missile(x_2)

$q' \leftarrow$ SUBST$(y_1/x_2$, west$/x_1$, Missile$(x_2)\}$.

Replacing / substituting.

~~$q' \leftarrow$ SUBST~~ $q' \leftarrow$ Missile(y_1)

There in KB for the match of q' it is found in KB, Rule 3 $\Rightarrow$ Missile(M_1) / 2.

$q \leftarrow$ Missile(M_1)

$\theta' \leftarrow$ UNIFY(q, q')

UNIFY(Missile(M_1), Missile(y_1))

L1 L2 L2/L1

Unification, the predicates are same,

$\theta' \leftarrow y_1/M_1$

newgoals $\leftarrow$ [P1 ... Pn || REST(goals)]

REST(goals) $\leftarrow$ sells$(x_1, y_1, z_1) \wedge$ Hostile(z_1)

$P \leftarrow$ NULL

newgoals ← [sells(x1, y1, z1) ∧ Hostile(z1)]

answers ← FOL-BC-ASK(KB, newgoals, compose(θ', θ))

FOL-BC-ASK(KB, sells(x1, y1, z1) ∧ Hostile(z1), compose(y1/M1, y1/x2, west/x1))

BC-ASK (KB, sells(x1, y1, z1) ∧ Hostile(z1), { y1/M1, y1/x2, west/x1})

Fifth Pass

since the goal is not empty.

goals = sells(x1, y1, z1) ∧ Hostile(z1), θ = { y1/M1, y1/x2, west/x1}

q' ← SUBST(θ, First(goals))

since the whole fact is not available in the KB, split the goal.

First(goals) ← sells(x1, y1, z1)

q' ← SUBST(y1/M1, y1/x2, west/x1, sells(x1, y1, z1))

q' ← sells(west, M1, z1)

check in the KB for the match, it is found in Rule 4 ⇒ $\frac{\text{missile}(x_3) \wedge \text{owns}(\text{Nono}, x_3)}{p \quad \downarrow}$

$\frac{\text{sells}(\text{west}, x_3, \text{Nono})}{q}$.

θ' ← UNIFY(q, q')

UNIFY (sells(west, x, Nono), sells(west, M1, z1)

~~θ' ← { M1/x3, z1/Nono }~~ { Nono/z1 }

θ' ← { M1/x3, Nono/z1} .

Now, newgoals ← [P1 ... Pn || REST(goals)]

Here P1 ... Pn ⇐ Missile(x3) ∧ owns(Nono, x3)

REST(goals) ⇐ Hostile(z1)

newgoals ← Missile(x3) ∧ owns(Nono, x3) ∧ Hostile(z1)

answers ← FOL-BC-ASK(KB, newgoals, compose(θ', θ)).

FOL-BC-ASK(KB, Missile(x3) ∧ owns(Nono, x3) ∧ Hostile(z1),

{ M1/x3, Nono/z1, y1/M1, y1/x2,

west/x1})

sixth pass

Again the goal is not empty.

goals = $Missile(x_3) \wedge owns(Nono, x_3) \wedge Hostile(z_1)$

$\theta = \{M_1/x_3, Nono/z_1, y_1/M_1, y_1/x_2, West/x_1\}$.

$q' \leftarrow SUBST(\theta, First(goals))$

since the whole fact is not available in KB, split the goals.

$First(goals) \leftarrow Missile(x_3)$

$q' \leftarrow SUBST(M_1/x_3, Nono/z_1, y_1/M_1, y_1/x_2, West/x_1, Missile(x_3))$.

$q' \leftarrow Missile(M_1)$

check in the KB for the match, it is found in KB Rule no 3

$\Rightarrow \underline{Missile(M_1)}$
q

$\theta' \leftarrow UNIFY(q, q')$

$\leftarrow UNIFY(Missile(M_1), Missile(M_1))$

$\theta' \leftarrow \{\}$.

$Nono, newgoals \leftarrow [P_1 \ldots P_n \,||\, REST(goals)]$

Here $P_1 \ldots P_n$ = NULL

$REST(goals) = owns(Nono, x_3) \wedge Hostile(z_1)$

$newgoals \leftarrow owns(Nono, x_2) \wedge Hostile(z_1)$

$answers \leftarrow FOL\_BC\_ASK(KB, owns(Nono, x_3) \wedge Hostile(z_1),$
$\{M_1/x_3, Nono/z_1, y_1/M_1, y_1/x_2, West/x_1\})$

seventh pass.

Again the goal is not empty

goals = $owns(Nono, x_3) \wedge Hostile(z_1)$

$\theta = \{M_1/x_3, Nono/z_1, y_1/M_1, y_1/x_2, West/x_1\}$.

$q' \leftarrow SUBST(\theta, First(goals))$

$First(goals) \leftarrow owns(Nono, x_3)$

After substitution $q' \leftarrow owns(Nono, M_1)$

Check for the match in KB, it is found in Rule ② ⇒ owns(Nono, M_1)
q

$\theta' \leftarrow$ UNIFY(q, q')

UNIFY(owns(Nono, M_1), owns(Nono, M_1))

$\theta' \leftarrow \{\}$

Now, newgoals $\leftarrow$ [$P_1 \ldots P_n$ || REST(goals)]

Here $P_1 \ldots P_n$ = NULL

REST(goals) = Hostile(z_1)

newgoals $\leftarrow$ Hostile(z_1)

answers $\leftarrow$ FOL_BC_ASK(KB, Hostile(z_1),

$\{M_1/x_3$, Nono/z_1, y_1/M_1, y_1/x_2, west/$x_1\}$

Eighth pass

Again goal is not empty

goals = Hostile(z_1)

$\theta = \{M_1/x_3$, Nono/z_1, y_1/M_1, y_1/x_2, west/$x_1\}$.

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

First(goals) $\leftarrow$ Hostile(z_1)

After substitution

$q' \leftarrow$ Hostile(Nono).

Check for the match in KB, it is found in Rule 6 ⇒
Enemy(x_4, America) → Hostile(x_4)
P q

$\theta' \leftarrow$ UNIFY(q, q')

UNIFY(Hostile(x_4), Hostile(Nono))

$\theta' \leftarrow$ Nono/x_4

Now, Newgoals $\leftarrow$ [$P_1 \ldots P_n$ || REST(goals)]

Here $P_1 \ldots P_n$ = Enemy(x_4, America)

q

REST(goals) = {}.(NULL)

Newgoals ← Enemy(x4, America)

answers ← FOL-BC-ASK(KB, Enemy(x4, America),
{M1/x3, Nono/z1, y1/M1, y1/x2, West/x1, Nono/x4}.

Nineth Pass

Again goal is not Empty.

goals = Enemy(x4, America)

θ = {M1/x3, Nono/z1, y1/M1, y1/x2, West/x1, Nono/x4}.

q' ← SUBST(θ, First(goals))

First(goals) ← Enemy(x4, America)

After substitution, q' ← Enemy(Nono, America)

Check for the match in KB, It is found in Rule 8 => Enemy(Nono, America)
q

θ' ← UNIFY(q, q')

— UNIFY(Enemy(Nono, America), Enemy(Nono, America))

All the predicates and arguments are same,

θ' ← {}

Now, newgoals ← [P1 ... Pn || REST(goals)]

P1 ... Pn = NULL

REST(goals) = NULL

newgoals ← {}

answers ← FOL-BC-ASK(KB, {}, {M1/x3, Nono/z1, y1/M1, y1/x2, West/x1, Nono/x4})

Tenth Pass

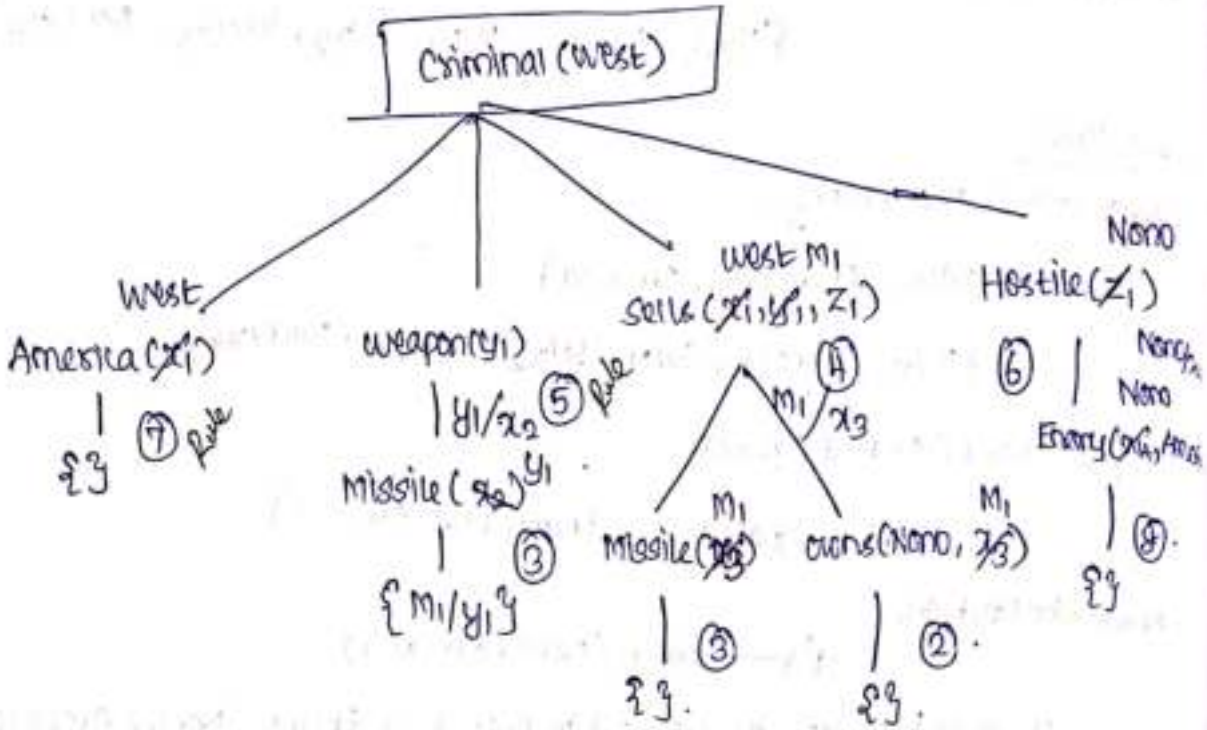
goals = {}.

By, Algorithm, IF goals is Empty, then return θ.

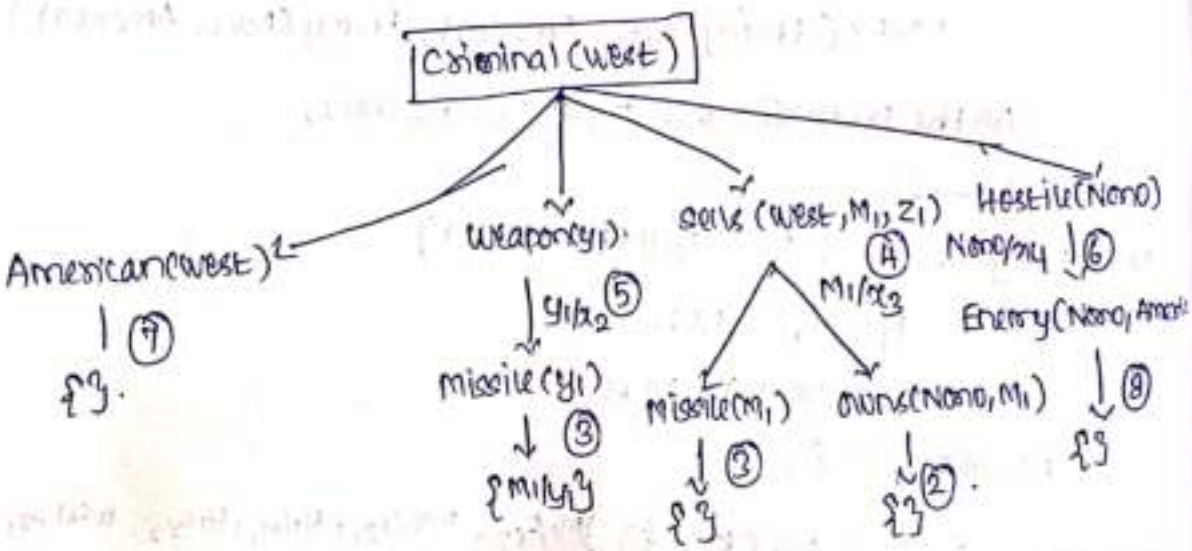
θ ← {M1/x3, Nono/z1, y1/M1, y1/x2, West/x1, Nono/x4}.

Prove that: Criminal(West)

$\theta = \{ West/x_1, M_1/y_1, Nono/z_1 \}$.



By Backward Chaining Algorithm, it is infered that "West is a criminal".



The Rules are represented in circle.

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 Colonel West, who is American"

Prove that "West is a criminal"

By Resolution Method,

Represent the Facts

1. "... It is a crime for an American to sell weapons to hostile nations"
2. " Nono ... has some missiles"
3. "All of its missiles were sold to it by Colonel West

We also need to know that missiles are weapons.

4. Missiles are weapons
6. We must also know that enemy of America counts as "hostile"
5. An Enemy of America counts as "Hostile"
6. West, who is American
7. The country Nono, an enemy of America

There are seven Facts From the Problem definition Provided.

Now, convert all Facts into its equivalent predicate in First Order Logic (FOL)

Step1: Converting Facts to FOL

1. "... It is a crime for an American to sell weapons to hostile nation"

FOL: American(x) ∧ weapon(y) ∧ sell(x, y, z) ∧ Hostile(z) → criminal(x)

2. "Nono ... has some missiles" (or) Nono owns some missiles

FOL: It has the term some, so use ∃ "(Existential Quantifier")

∃(x): Missile(x) ∧ owns(Nono, x)

3. All of its missiles were sold to it by colonel west

FOL → It has the term All, so use ∀ (Universal Quantifier)

∀(x): Missile(x) → sells(West, x)

(But in the given problem statement it has the conjunction word Nono, has some missile, so, the FOL is written as

FOL: ∀(x): Missile(x) ∧ owns(Nono, x) → sells(West, x, Nono)

4. Missiles are weapons

FOL: missile(x) → weapon(x)

5. An Enemy of America counts as "Hostile"

FOL: Enemy(x, America) → Hostile(x)

6. West, who is American

FOL: American(West)

7. The country Nono, an Enemy of America

Enemy(Nono, America)

First order logic

1. American(x) ∧ weapon(y) ∧ sells(x, y, z) ∧ Hostile(z) → criminal(x)
2. ∃(x): owns(Nono, x) ∧ missile(x)
3. ∀(x): missile(x) ∧ owns(Nono, x) → sells(West, x, Nono)
4. missile(x) → weapon(x)
5. Enemy(x, America) → Hostile(x)
6. American(West)
7. Enemy(Nono, America)

Step 2: Now convert the above FOL into CNF (Conjunctive Normal Form)

1. American(x) ∧ weapon(y) ∧ sells(x, y, z) ∧ Hostile(z) → criminal(x)
   (a) (b)

(Now, in the above FOL, there exist a implication (→), we need to remove →. we know that

$$a \rightarrow b = \neg a \vee b$$

$$\neg[\text{American}(x) \wedge \text{weapon}(y) \wedge \text{sells}(x, y, z) \wedge \text{Hostile}(z)] \vee \text{criminal}(x)$$

CNF $\neg\text{American}(x) \vee \neg\text{weapon}(y) \vee \neg\text{sells}(x, y, z) \vee \neg\text{Hostile}(z) \vee \text{criminal}(x)$

2. ∃(x): owns(Nono, x) ∧ missile(x)

To eliminate the existential quantifier replace ∃ variable (here the ∃ variable is x) either by special constants known as skolem constants by introducing a new constant M_1.

∃(x): owns(Nono, M_1) ∧ missile(M_1)

owns(Nono, M_1) ∧ missile(M_1) [separating the clause)

CNF: i) owns(Nono, M_1)

ii) missile(M_1)

3. $\forall(x)$: missile(x) $\wedge$ owns(Nono, x) $\rightarrow$ sells(west, x, Nono)

[Drop the Universal Quantifier]

missile(x) $\wedge$ owns(Nono, x) $\rightarrow$ sells(west, x, Nono)

(a) (b)

[Here an implication exists, remove it by $a \rightarrow b = \neg a \vee b$]

$\neg$[missile(x) $\wedge$ owns(Nono, x)] $\vee$ sells(west, x, Nono)

CNF: $\neg$missile(x) $\vee$ $\neg$owns(Nono, x) $\vee$ sells(west, x, Nono)

4. missile(x) $\rightarrow$ weapon(x)

(a) (b)

[Here an implication exists, replace it by $a \rightarrow b = \neg a \vee b$]

CNF: $\neg$Missile(x) $\vee$ weapon(x)

5. Enemy(x, America) $\rightarrow$ Hostile(x)

(a) (b)

[Here exist an implication, replace it by $a \rightarrow b = \neg a \vee b$]

CNF: $\neg$Enemy(x, America) $\vee$ Hostile(x)

6. American(west)

[Nothing to do all the terms are correct]

CNF: American(west)

7. Enemy(Nono, America)

[Nothing to do all the terms are correct]

CNF: Enemy(Nono, America)

Conjunctive Normal Form

1. ¬ American(x) ∨ ¬ weapon(y) ∨ ¬ sells(x, y, z) ∨ ¬ Hostile(z) ∨ Criminal(x)
2. owns(Nono, M1)
3. missile(M1)
4. ¬ Missile(x) ∨ ¬ owns(Nono, x) ∨ sells(West, x, Nono)
5. ¬ missile(x) ∨ weapon(x)
6. ¬ Enemy(x, America) ∨ Hostile(x)
7. American(West)
8. Enemy(Nono, America)

Step3: Negate the conclusion

Conclusion: West is a criminal)

FOL: Criminal(West)

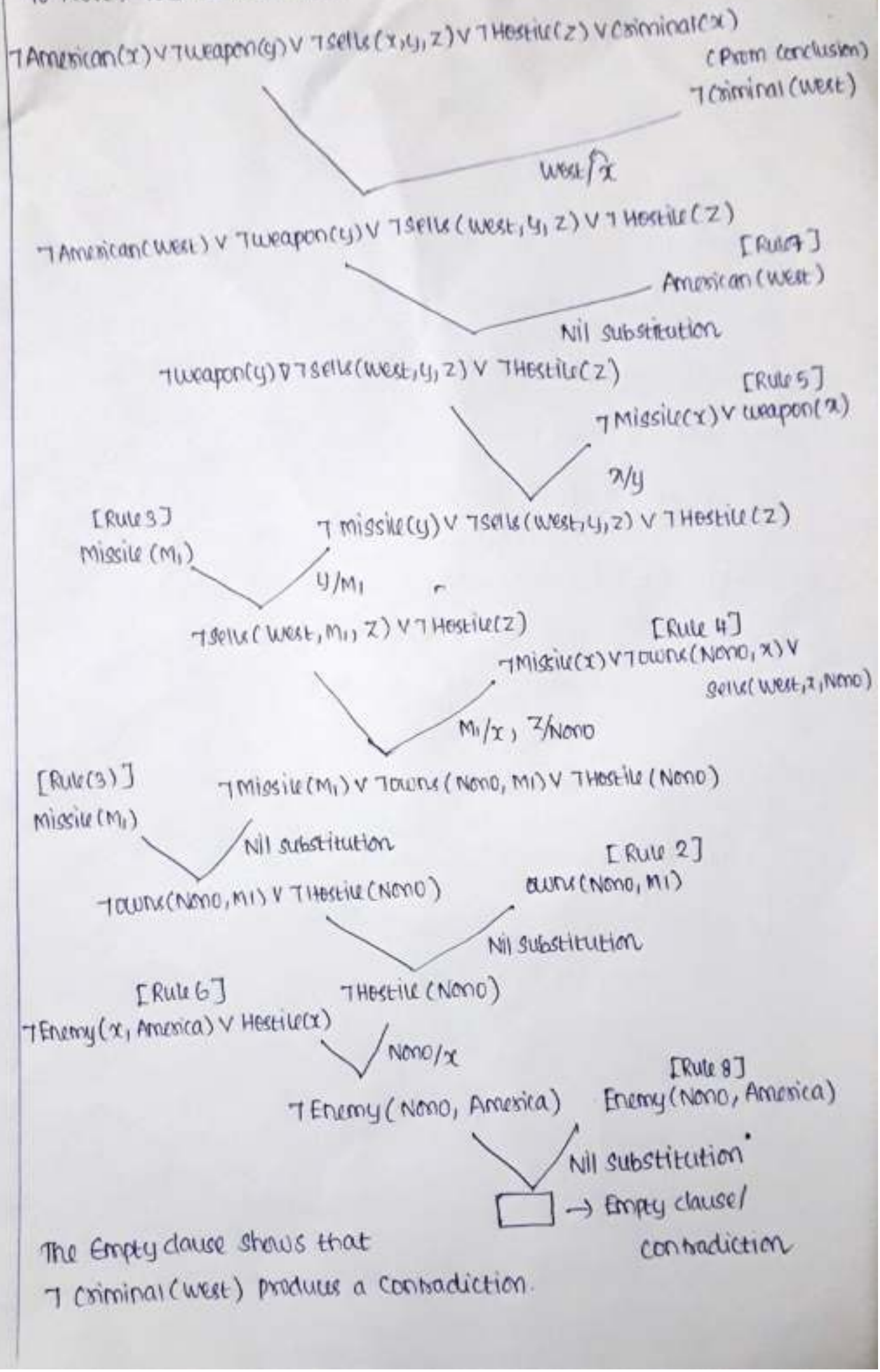
CNF: Criminal(West)

Negation: ¬Criminal(West)

Step4: Resolve using a Resolution Tree

By using Unification Algorithm

To Prove: West is a criminal



The Empty clause shows that ¬Criminal(West) produces a contradiction.

By Forward chaining,

First write the definition of Forward chaining, Followed by the algorithm, then you can proceed with problem solving. In Forward chaining, the left-side matches the root node and the right side to create a new node.

First convert the given facts into its equivalent FOL (or) Predicate Logic. Here we use the same example to prove "West is Criminal"

FOL: [Don't worry about implications, just remove Universal Quantifier and apply skolem constant to remove Existential Quantifier]

Knowledge Base:

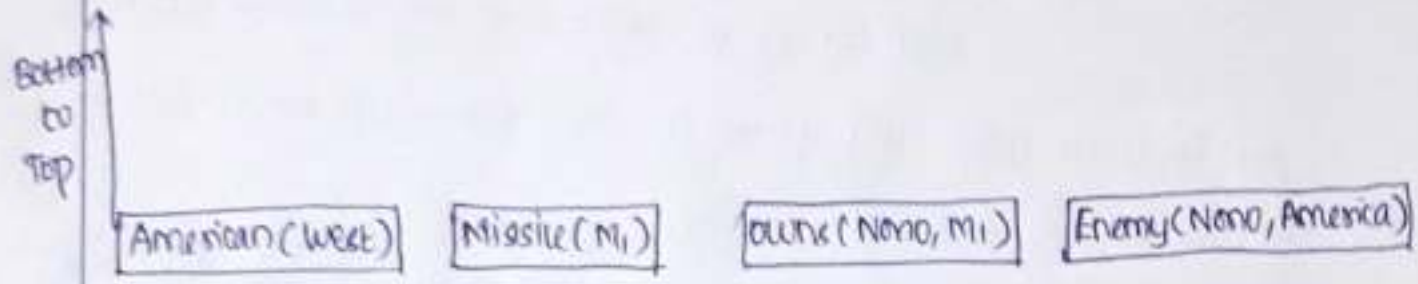
1. American(x) ∧ weapon(y) ∧ sells(x,y,z) ∧ Hostile(z) → Criminal(x)
2. owns(Nono, M1)
3. Missile(M1)
4. Missile(x) ∧ owns(Nono, x) → sells(West, x, Nono)
5. Missile(x) → weapon(x)
6. Enemy(x, America) → Hostile(x)
7. American(West)
8. Enemy(Nono, America)

[You can trace the Algorithm step by step (or) you can draw the proof directly, to get more marks just trace atleast 2 iterations as per Algorithm, then draw the proof].

For step by step Iteration Refer Notes.

Step1:

As it is Forward chaining, start from the initial state, Choose the statement, that doesn't have Implication [Here Rule 2, 3, 7, 8 have no Implication]



Step2:

Now, Check for q where p matches → statement should be in the Form P→q

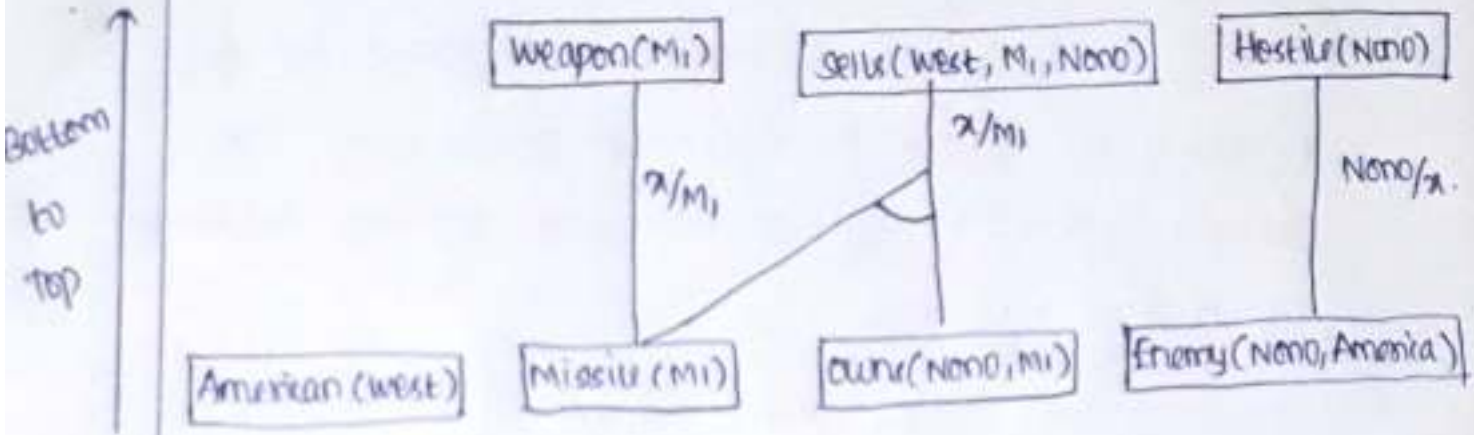
[Consider the Rule1, here American predicate matches, and the also it has single argument, but the second ~~statement~~ Predicate weapon does not match so Rule 1 cant be used]

[From Rule 4 ⇒ $\underbrace{Missile(x) \wedge Owns(Nono, M_1)}_{P} \rightarrow \underbrace{Sells(West, x, Nono)}_{q}$

[From Rule 5 ⇒ $\underbrace{Missile(x)}_{P} \rightarrow \underbrace{Weapon(x)}_{q}$

[From Rule 6 ⇒ $\underbrace{Enemy(x, America)}_{P} \rightarrow \underbrace{Hostile(x)}_{q}$

Now ~~draw~~ the Forward chaining proof



Step 3:

Now, Again check for q where p matches, Now the statement to be matched is

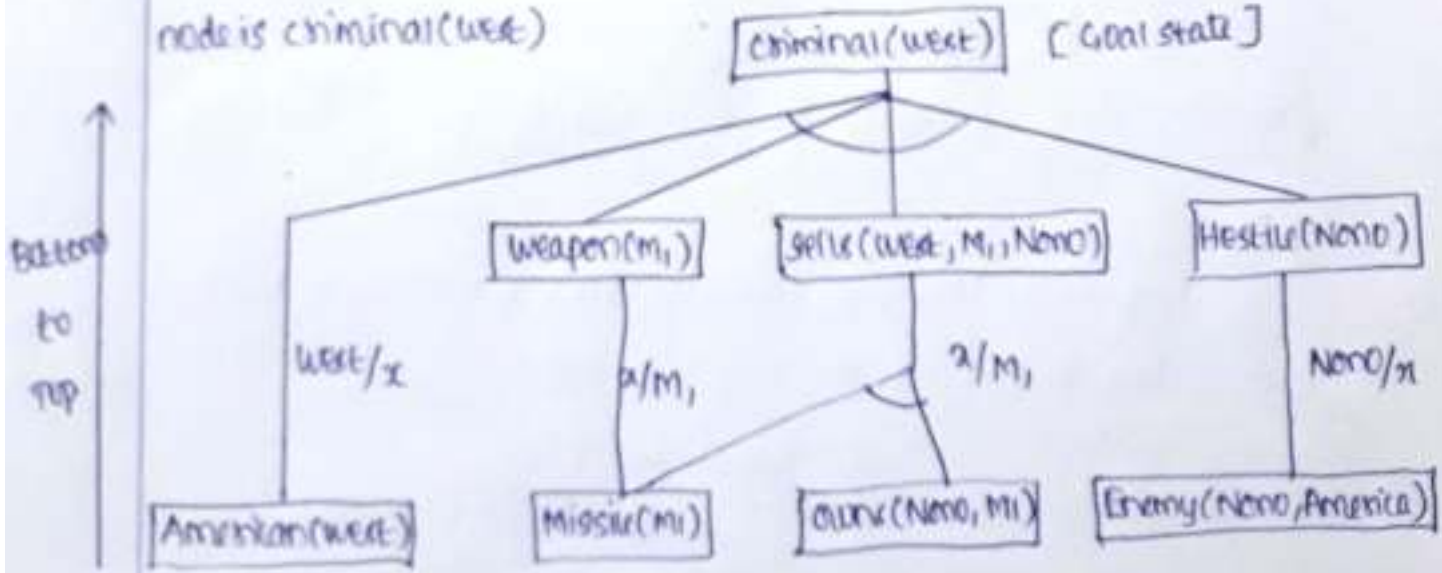
American(west) ∧ weapon(M_1) ∧ Sells(west, M_1, Nono) ∧ Hostile(Nono)

check in the Knowledge Base (or) Rule list, it matches with the Rule 1

American(x) ∧ weapon(y) ∧ Sells(x, y, z) ∧ Hostile(z) → Criminal(x)

p q

Here p matches (with predicate and argument), then the new node is criminal(west)



Initial state

The Proof tree generated by Forward chaining on the crime Example. The initial facts appear at the bottom level, facts inferred on the First iteration in the middle level & facts inferred on the second iteration at the top level.

By Backward chaining,

Write the definition of Backward chaining,

Followed by the algorithm, then proceed with problem solving. In Backward chaining, the right side matches the root node and the left side to create a new node.

Same Example "West is criminal"

Considering the Knowledge Base used in Forward chaining.

Step1: In Backward chaining, start from the goal state.

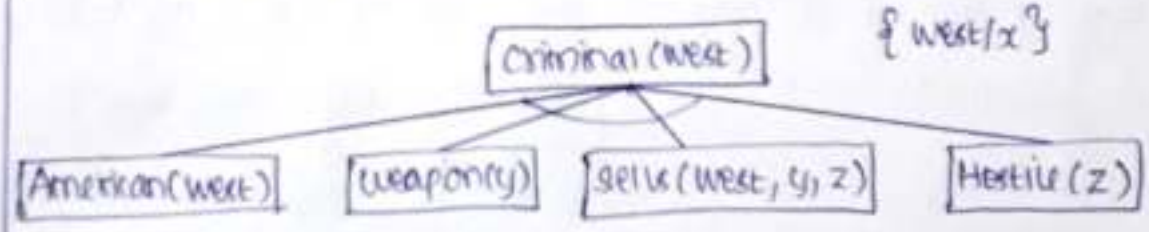
criminal(West)

Check for the predicate criminal(West) in the knowledge Base, in Rule 1 it matches.

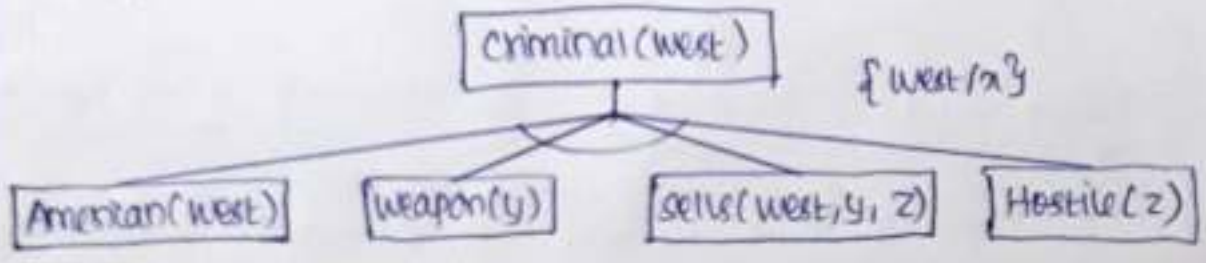
American(x) ∧ weapon(y) ∧ sells(x,y,z) ∧ Hostile(z) → criminal(x)

P q

Here q matches and P will be the new node.



Step2: Check for the predicate American(West), it matches with the Rule 7, but it does not give any new node.



{} → Nil substitution

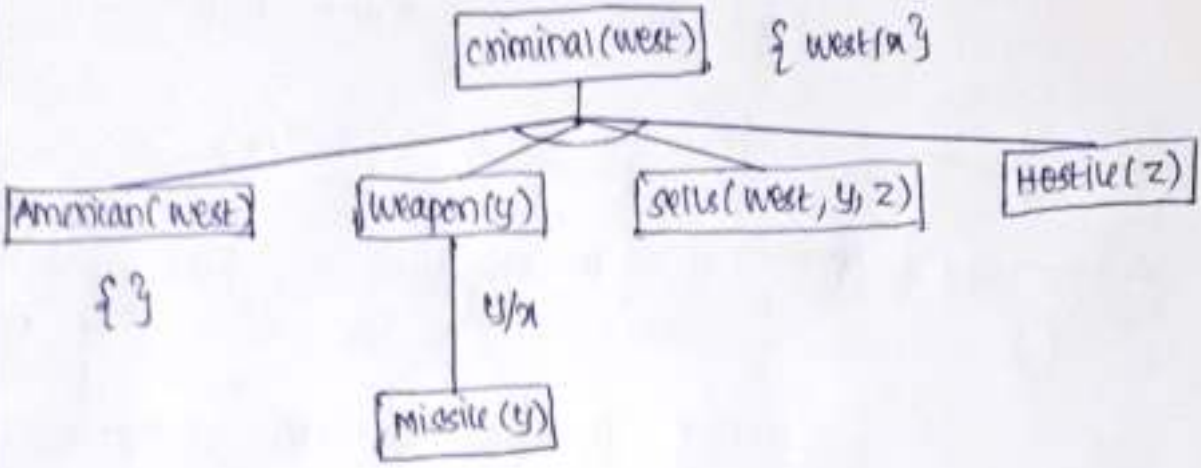
Step3:

Check for the next predicate weapon(y), it matches with the Rule 5, it gives a new node

Rule 5 => Missile(x) → weapon(y)

P — new node

q — matched node



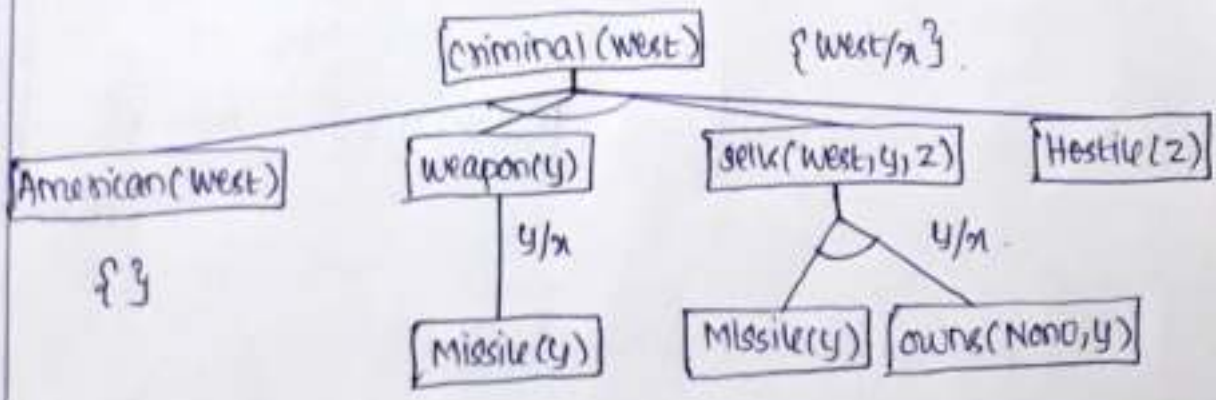
Step4:

Check for the next predicate sells(west, y, z), it matches with the Rule 4, it gives a new node.

Rule 4 => Missile(x) ∧ owns(Nono, z) → sells(west, x, Nono)

P — new node

q — matched node



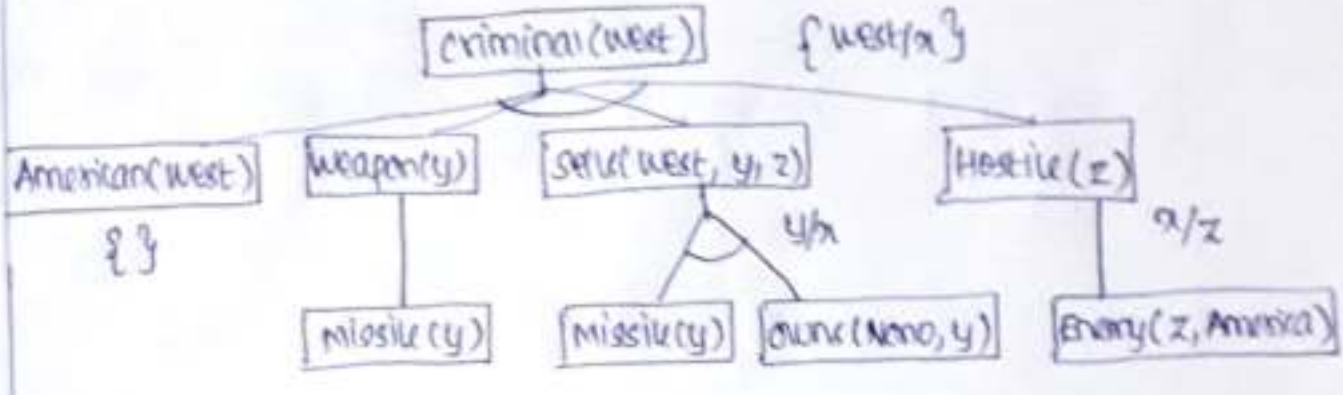
Step5:

Check for the next predicate Hostile(z), it matches with the Rule 6, it gives a new node

Rule 6 ⇒ Enemy(x, America) → Hostile(x)

P — new node

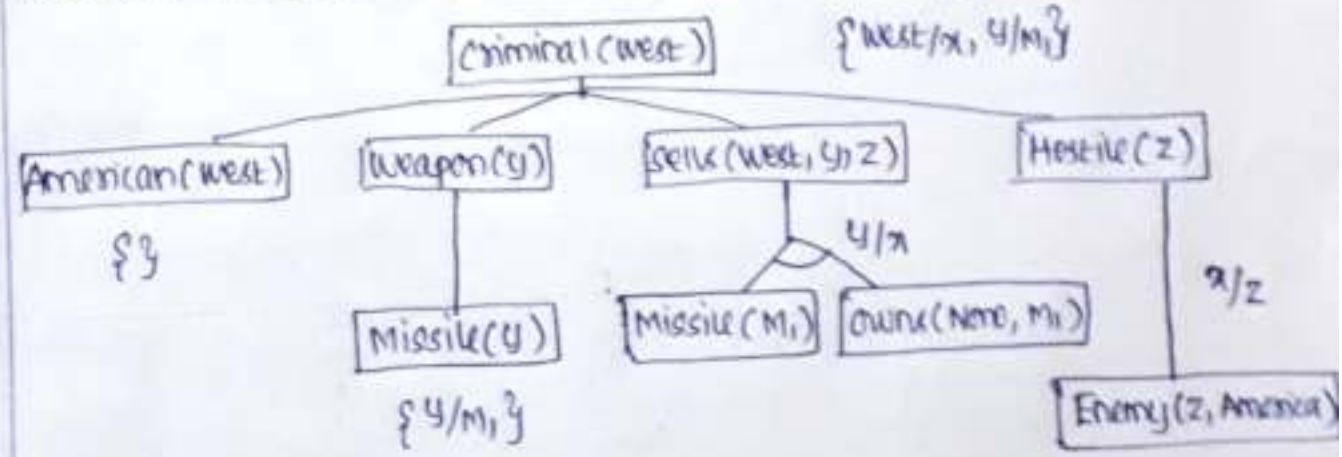
q — matched node



Step6:

Now, move to the terminal node,

Consider Missile(y) → It matches with Rule 3 replacing y/M1

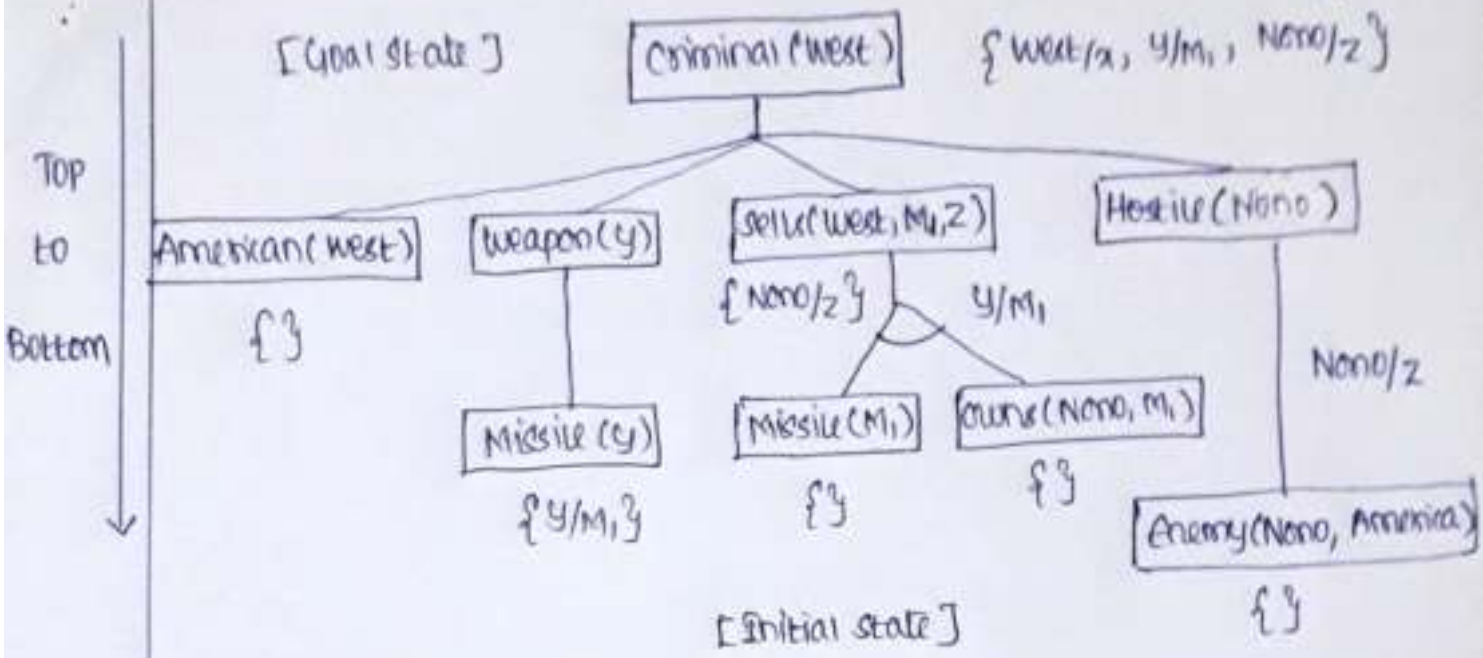


Again consider the other predicates.

Missile(M1) matches with Rule 3 without substitution (or) Nil substitution

Owns(Nono, M1) matches with Rule 2 Nil substitution

Enemy(z, America) matches with Rule 8 replacing Nono/z.



The proof tree generated by Backward chaining on the crime Example to prove West is a criminal. The tree should be read depth first, left to right.