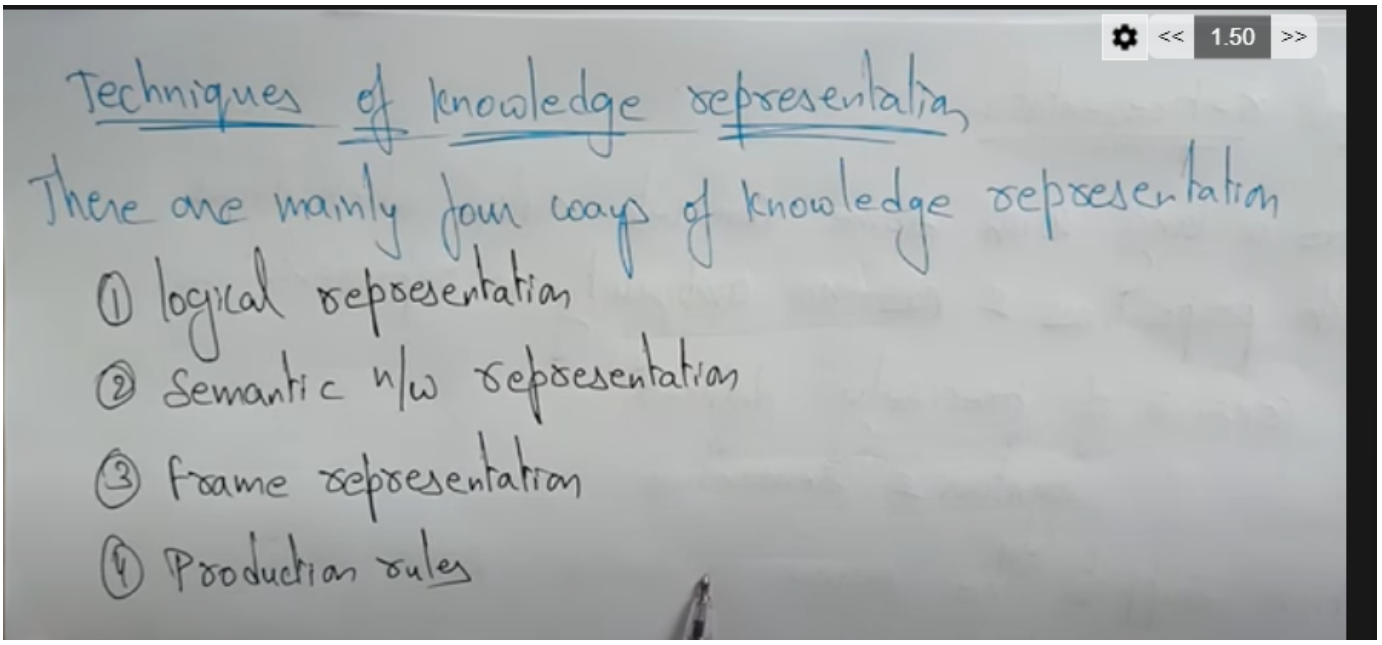
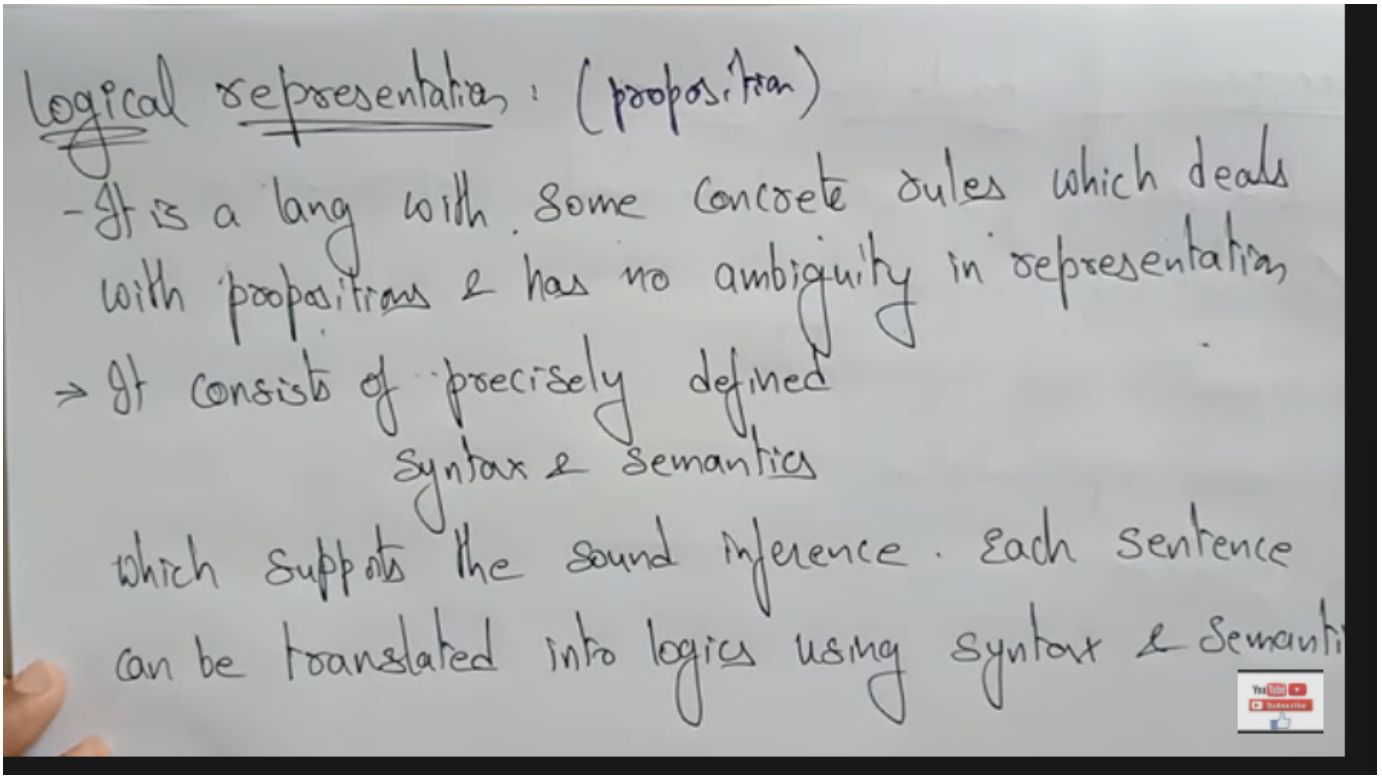
Unit 3

KNOWLEDGE AND REASONING:

Reasoning -> processing of knowledge





- Syntax: defines well-formed sentence in the l
- Semantics: defines the truth of meaning of sentence in a world.

Logical Representation

→ Propositional logic
→ First order predicate logic

① Propositional logic : (PL)

→ PL is the

in a world.

Logical Representation

→ Propositional logic ✓
→ First order predicate logic ✓

① Propositional logic : (PL)

→ PL is the simplest logic ✓
→ A proposition is a declarative statement that's either T/F

⇒ Propositional logic cannot predicate, it can say true or false.

⇒ Connectives

word	symbol	example
→ Not	¬	¬A
→ and	∧	A∨B
→ OR	∨	A∧B
→ implies	→	A→B
→ if and only if (biconditional stmt)	↔	A↔B

A — It is hot

B — It is humid

C — It is raining

Conditrary:

- If it is humid, then it is hot ⇒ Prepositionlogic
  B → A ⇒ B → A
- If it is hot and humid then it is not raining
  A ∧ B → ¬C = (A∧B) → ¬C

So, Proposition is a statement of a fact

Limitations of propositional logic

⇒ We cannot represent relations like ALL, some & none with propositional logic

a. All the girls are intelligent
b. some apples are sweet.

} not declarative stmt

⇒ Propositional logic has limited expressive power

⇒ In PL, we cannot describe statements in terms of their properties & logical relationships

Knowledge base for Wumpus world

Atomic Propositian Variables for Wumpus world

1. Let $P_{i,j}$ be true if there is a pit in the room $[i,j]$
2. let $B_{i,j}$ be true if agent perceives breeze in $[i,j]$
3. Let $W_{i,j}$ be true if there is Wumpus in square $[i,j]$
4. Let $S_{i,j}$ be true if agent perceives stench in $[i,j]$
5. Let $V_{i,j}$ be true if that square $[i,j]$ is visited
6. Let $G_{i,j}$ be true if there is gold

(5) let $V_{i,j}$ be true if that square $[i,j]$ is visited

(6) let $G_{i,j}$ be true if there is gold

(7) let $OK_{i,j}$ be true if room is safe

$R_1 \Rightarrow \neg S_{11} \rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{21}$

$R_2 \Rightarrow \neg S_{21} \rightarrow \neg W_{11} \wedge \neg W_{21} \wedge \neg W_{22} \wedge \neg W_{31}$

$R_3 \Rightarrow \neg S_{12} \Rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{22} \wedge \neg W_{13}$

$R_4 \Rightarrow S_{12} \rightarrow W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$

1,4 S	3,4	2,4 B	4,4 PIT
1,3 W	2,3 S, B G	3,3 PIT	4,3 B
1,2 S	3,2	3,2 B	4,2
1,1	2,1	3,1	4,1

$R_1 \Rightarrow \neg S_{11} \rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{21}$

$R_2 \Rightarrow \neg S_{21} \rightarrow \neg W_{11} \wedge \neg W_{21} \wedge \neg W_{22} \wedge \neg W_{31}$

$R_3 \Rightarrow \neg S_{12} \Rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{22} \wedge \neg W_{13}$

$R_4 \Rightarrow S_{12} \rightarrow W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$

1,4 S	3,4	2,4 B	4,4 P
1,3 W	2,3 S, B G	3,3 P	4,3 B
1,2 S	3,2	3,2 B	4,2
1,1		3,1 P	4,1 B

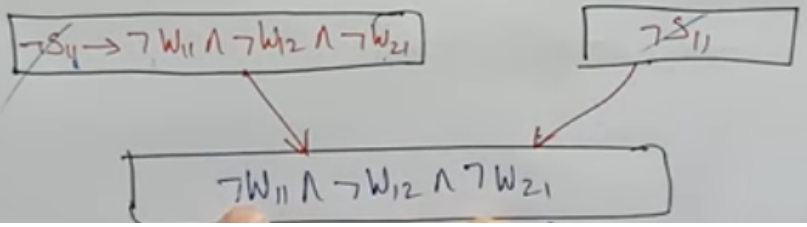
→ We can poove Wumpus is in the room (1,3) using propositional rules

① Apply Modus ponens with $\neg S_{11}$ & R_1:

⇒ We first apply MP rule with R_1 which is $\neg S_{11} \rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{21}$ & $\neg S_{11}$ which will gives the o/p $\neg W_{11} \wedge W_{12} \wedge W_{12}$

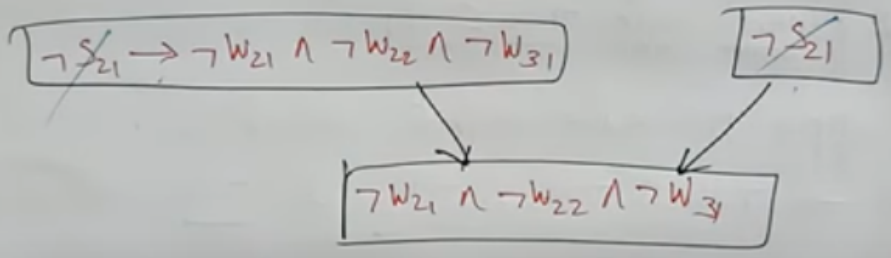
② Apply Modus ponens with $\neg S_{11}$ & R_1:

⇒ We first apply MP rule with R_1 which is $\neg S_{11} \rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{21}$ & $\neg S_{11}$ which will gives the o/p $\neg W_{11} \wedge W_{12} \wedge W_{12}$



$\neg W_{11}$, $\neg W_{12}$ & $\neg W_{21}$

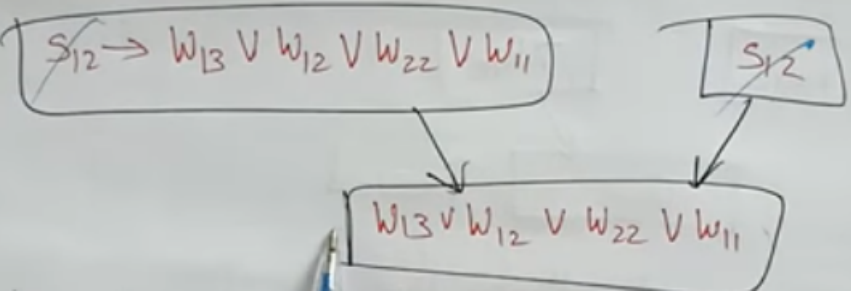
⇒ Apply Modus ponens to $\neg S_{21}$ & R_2:



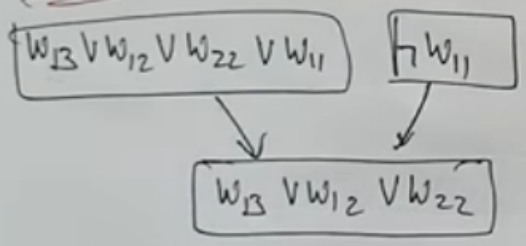
Apply & elimination rule we will get three statements

$\neg W_{21}$, $\neg W_{22}$, $\neg W_{31}$

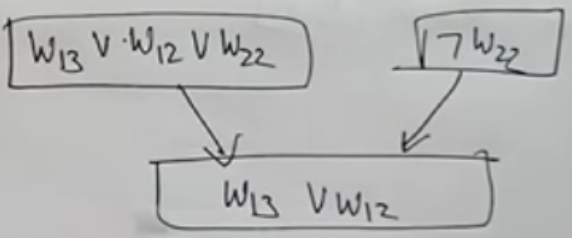
③ Apply Modus ponens to S_{12} & R_4:



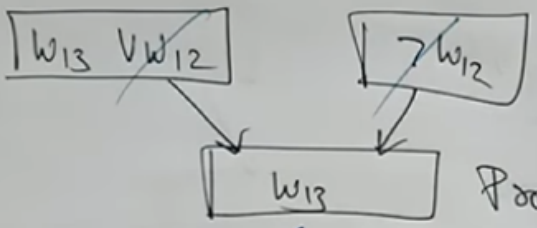
⇒ Apply unit resolution on



⇒ Apply unit resolution on



⇒ Apply unit resolution on $W_{13} \vee W_{12}$ & $\neg W_{12}$:



Proved

Hence it it proved that the wumpus is in the room [1,3]

First-Oder Logic in Artificial Intelligence

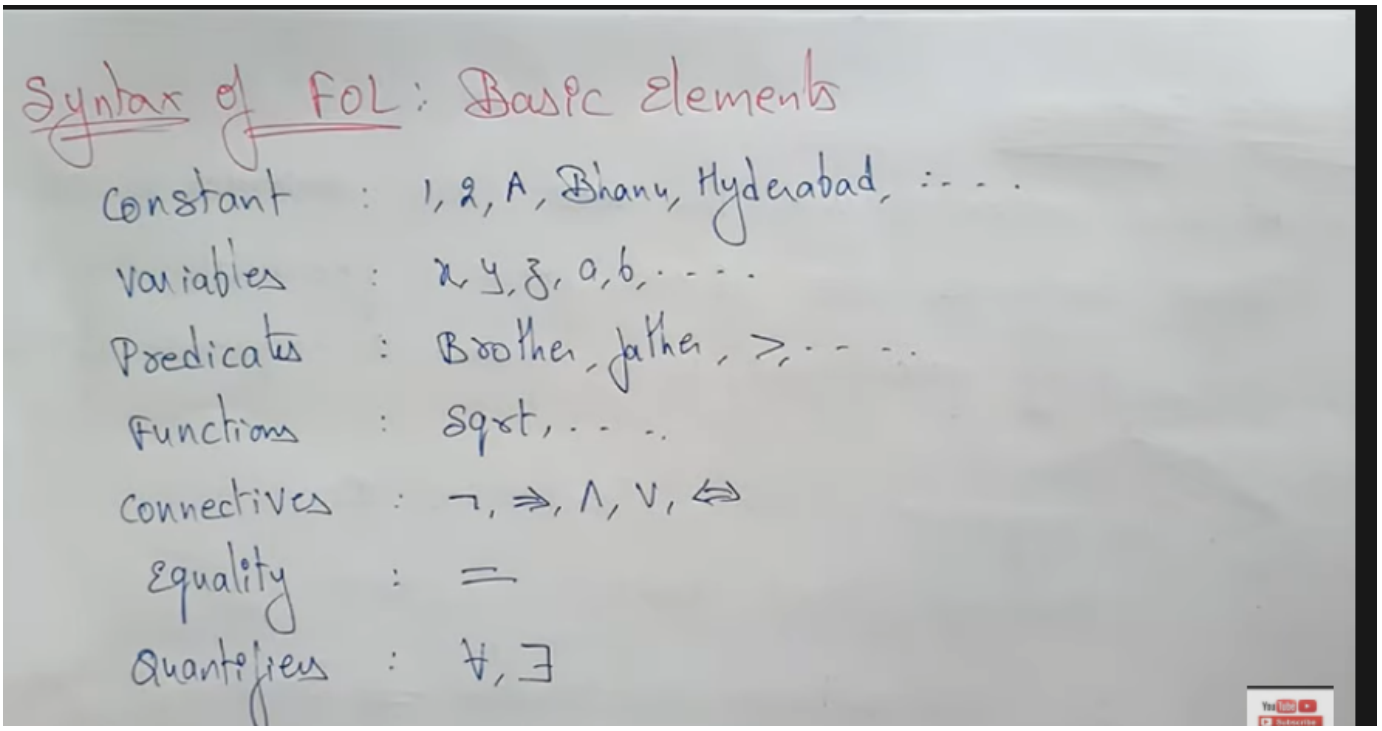
- FOL is another way of knowledge representation in A.I.
  It is an extension to PL. (Propositional Logic)
- FOL is also know nas Predicate logic. It is a powerful language that develops information about the objects in a mor easy way and can also express the relationship between those objects.
- FOL does not only assume that the world contains facts like PL but also assumes

- FOL does not only assume that the world contains facts like PL but also assumes

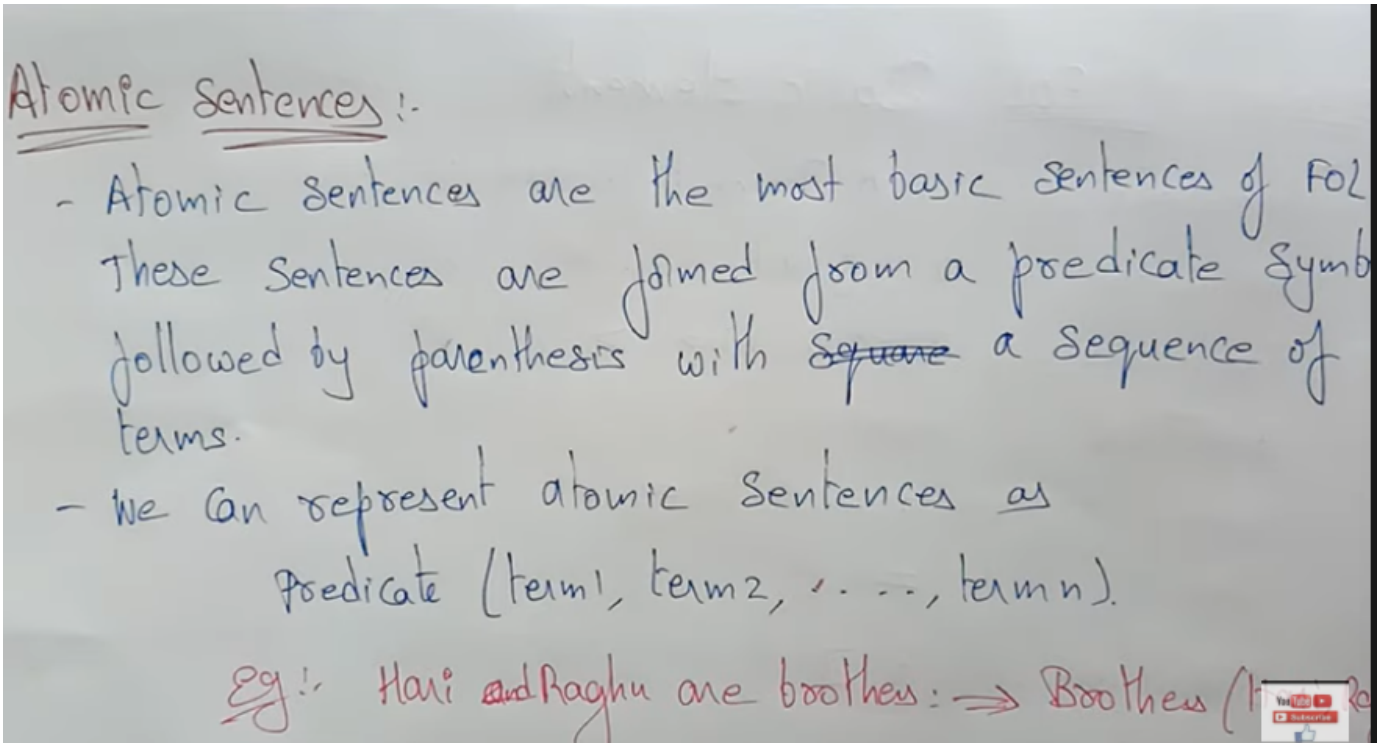
1. objects: A, B, people, no:, Colors, Wars, pits, Wumpus, .

2. Relations: It can be unary relation such as
   n- any relation such as: the sister of, brother of, has color.
   - Function: Father of, best friend, end of, ...

- As a natural lang, FOL has two main parts
  a. Syntax
  b. Semantics



fol->atomic and complex sentences



Hari and Raghu are brothers: ⇒ Brothers (Hari, Raghu)

Tommy is a dog: ⇒ dog (Tommy)

omplex Sentences

Predicate(term1, term2 ...)

- Complex Sentences are made by combining atomic Sentences using Connectives (¬, ∨, ∧, ⇒, ⇔)

irst order logic statements divided into 2 parts:-

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

Consider the stmt

stmt → "x is an integer"

It consists of two parts,

first part: x is the subject of stmt

second part: "is an integer" is known as predicate.

x is an Integer

Subject Predicate

Quantifiers in FOL :-

→ A Quantifier is a lang ele which generates quantification

→ these are the symbols that permit to determine or identify the range and scope of the variable in the logic expression.

There are two types of quantifier :-

a. Universal Quantifier (for all, everyone, everything) — range

b. Existential Quantifier (for some, atleast one) — scope

Universal Quantifier :

It is a symbol of logical representation which specifies that the statement within its range is true for everything or every instance of particular thing.

→ It is represented as by a symbol ∀.

(In UQ we use implication "→")

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

- for all x
- for each x

If x is a variable, the $\forall x$ is read as:

- for all x
- for each x
- for every x

rang

Existential Quantifier:

∀ → range is true

It is a type of Quantifier, which express that the stmt within its scope is true for at least one instance of something.

It is denoted by ∃.

{In Existential quantifier we always use AND or conjunction symbol (∧).}

⇒ If x is a variable, then existential quantifier will be $\exists(x)$ is read as:

It is denoted by ∃.

{In Existential quantifier we always use AND or conjunction symbol (∧).}

⇒ If x is a variable, then existential quantifier will be $\exists(x)$ is read as:

- There exists a 'x', for some 'x', for atleast one

Eg: {All man drink coffee.} ∀

- let x is variable.

. x_1 drink coffee ∧ x_2 drink coffee ....

∧ x_n drink coffee.

So, we can write this as,

$\forall x \; man(x) \rightarrow drink(x, coffee)$

There are all x where x is a man who drink coffee

(1) All birds fly. ∀

Predicate is "fly (bird)

$\forall x \; bird(x) \rightarrow fly(x)$

x-bird.

(2) Every man respects his parent ∀

Predicate is "respect (x, y)" where x = man, y = parent.

$\forall x \; man(x) \rightarrow respects(x, parent)$

Here every man, so we use ∀

Eg:- Some boys are intelligent. ∃

x_1 is intelligent ∨ x_2 is intelligent ..... ∨ x_n is intelligent

$\exists x : boy(x) \wedge intelligent(x)$

→ There are some ~~boy~~ x where x is a boy who is intelligent.

* The main connective for ∀ is '→'.
The main connective for ∃ is '∧'.

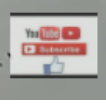
Inference in First order logic :-

→ Inference in FOL is used to ~~ded~~ deduce new facts Sentences from existing sentences.

Before understanding FOL inference rule, let's understand some basic terminologies used in FOL.

① Substitution

It is a fundamental operation performed on terms and formulas. It occurs in all inference sys in FOL.



① Substitution

It is a fundamental operation performed on terms and formulas. It occurs in all inference sys in FOL.

F[a/x] F[b/x]

Substitute a constant "a" in place of variable "x"

② Equality:-

FOL logic does not only use predicate & terms making atomic sentences but also uses another way, which is equality in FOL.

Eg:- Brother(John) = Smith

Here, the obj refered by Brother(John) is similar to obj refered by Smith. The equality symbol can also be used with negation to represent that two terms are not the same objects.

Eg:- ¬(x=y) which is equivalent to x≠y

FOL inference rules for quantifier:

As PL we also have inference rules in FOL.

- → Universal Generalization.
- → Universal Instantiation
- → Existential Instantiation
- → Existential Introduction.

① Universal Generalization:

It is a valid inference rule which states

① Universal Generalization:

It is a valid inference rule which states tha

if premise P(c) is true for any arbitrary element c in

universe of discourse, then we can have a conclusion as $\forall x P(x)$.

- It can be represented as, $\frac{P(c)}{\forall x P(x)}$

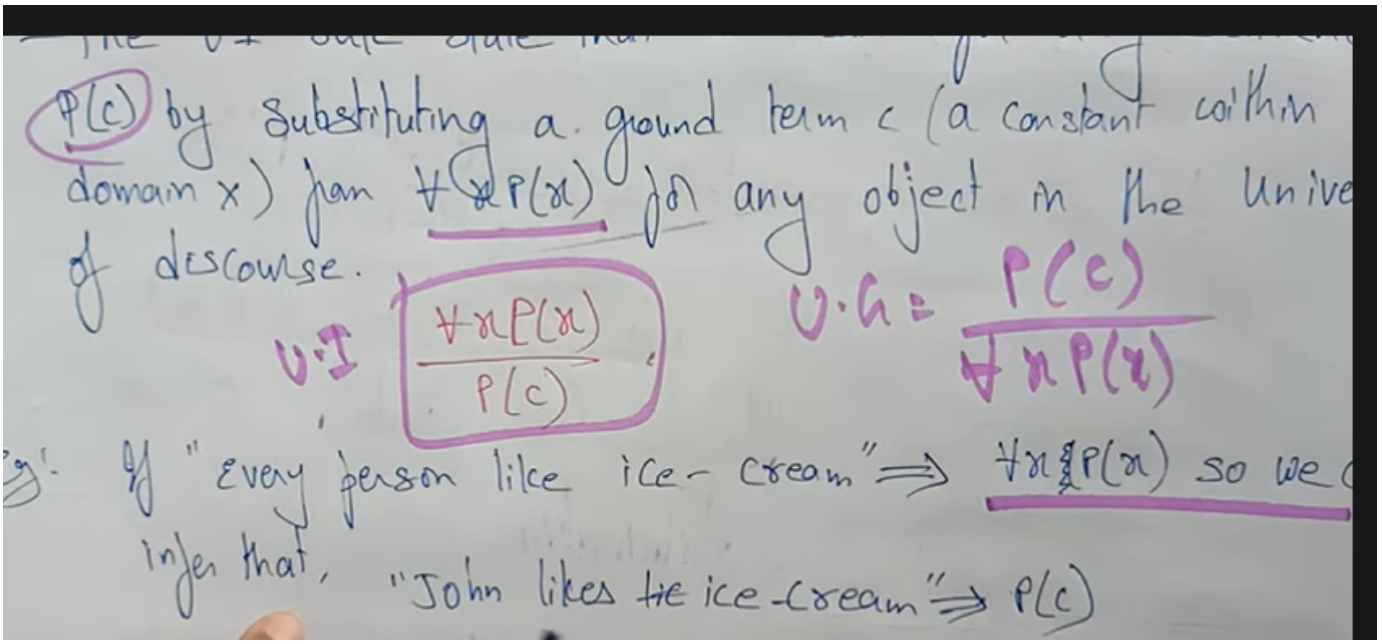
Eg:- Let's represent, P(c): "A byte contains 8 bits";

So, for $\forall x P(x)$ "All bytes contain 8 bits," it will also be true.

be true.

(2) Universal Instantiation = elimination

It is also called universal elimination. It can be applied multiple times to add new sentences.



Existential Instantiation :- = elimination

→ It is also called Existential Elimination, which is a valid inference rule in FOL.

→ It can be applied only once to replace the existential sentence

→ This rule states that one can infer P(c) from the formula given in the form $\exists x\, P(x)$ for a new constant symbol 'c'.

$$\frac{\exists x\, P(x)}{P(c)}$$

Existential introduction :-

→ It is also called as existential generalization.

→ This rule states that if there is some element c in the universe of discourse which has a property P, then we can infer that there exists something in the universe which has property P.

$$\frac{P(c)}{\exists x\, P(x)}$$

Eg :- "Pinky got good marks in ma

"Therefore, some one got goo

Unification

It is all about making the expressions look identical so, for the given expressions to make them look identical we need to do substitution.

eg:- $P(x, f(y))$ ① ; $P(a, f(g(z)))$ ②

$x = a$

$y = g(z)$

unification : $[a/x, g(z)/y]$

⇒ In the above eg, substitute x with a, & y with $g(z)$. it will be represented as a/x & $g(z)/y$.

⇒ with both the substitutions, the first expression will

identical to the second expression & the substitution set will be $[a/x, g(z)/y]$

Conditions for unification :-

① Predicate symbol must be same, atoms or expressio with different predicate symbol can never be unified.

② Number of arguments in both expressions must be ide

③ unification will fail if there are two similar variable present in same expression.

Unification Algorithm

⇒ Algorithm : Unify (L_1, L_2)

Step 1 :- If L_1 or L_2 is a variable or constant, then :

(a) If L_1 & L_2 are identical return NIL.

(b) Else if L_1 is a variable, then if L_1 occurs in L_2 then return FAIL, else return $\{(L_2/L_1)\}$

(c) Else if L_2 is a variable, then if L_2 occurs in L_1 then return FAIL, else return $\{(L_1/L_2)\}$

(d) Else return FAIL.

Step 2 :- If the initial predicate symbol in L_1 & L_2 are not identical, then return FAIL.

Step 3 :- If L_1 & L_2 have a different number of arguments, then return FAIL.

Step 4 : Set SUBST to NIL

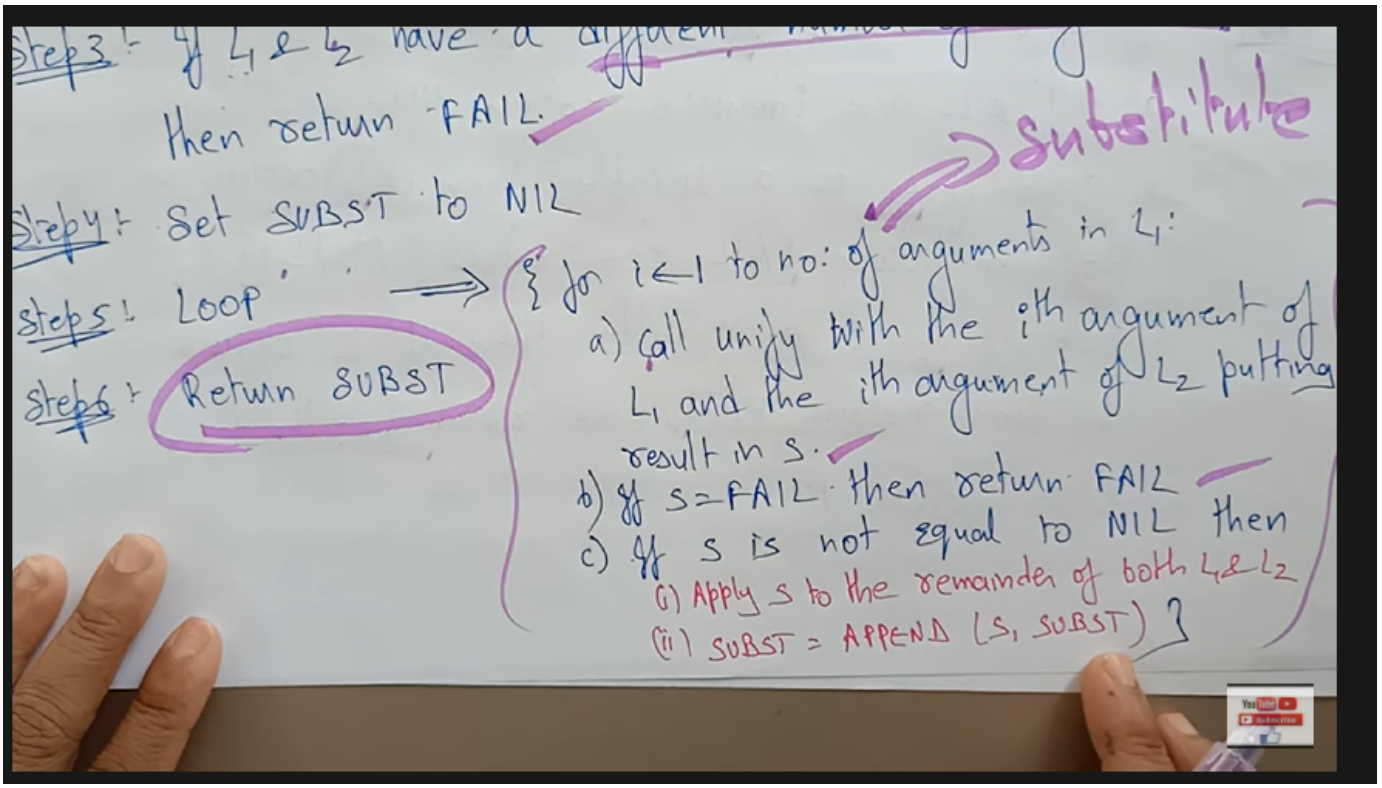
Step 5 : Loop ⟹ { for i ← 1 to no: of arguments in L_1 :

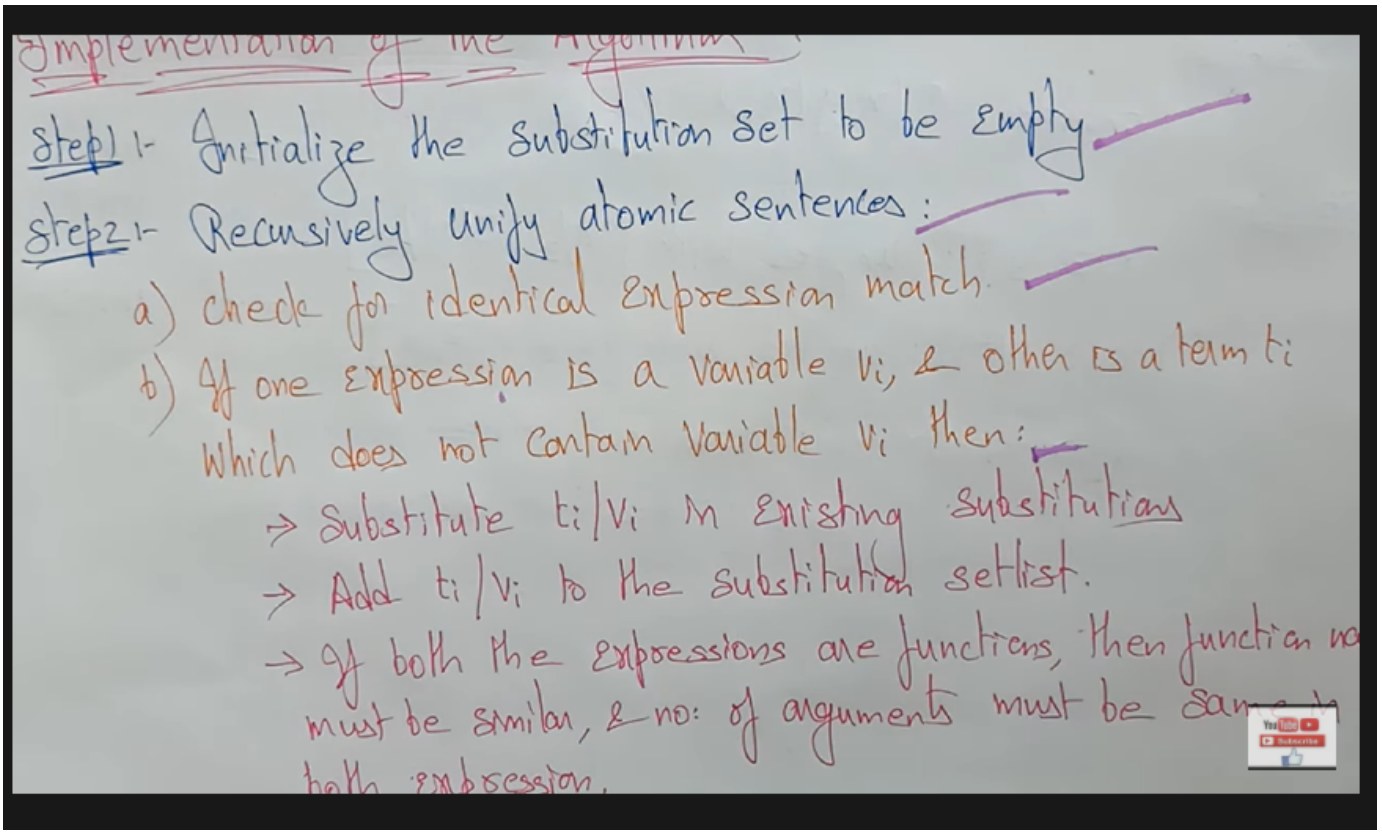
a) Call unify with the ith argument of L_1 and the ith argument of L_2 putting result in S.

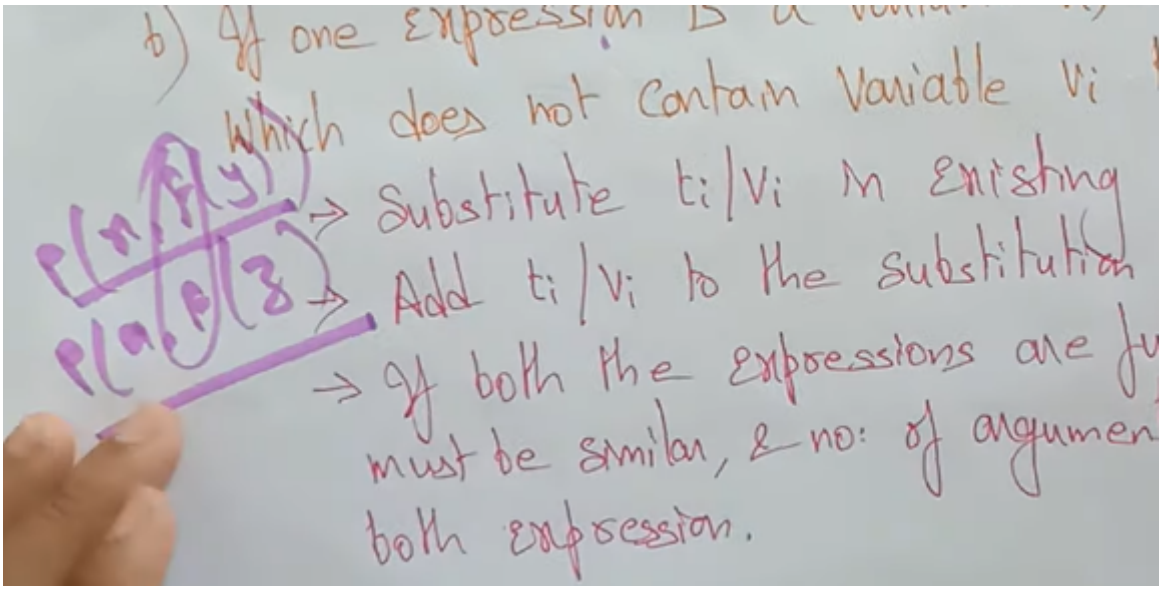
b) If S = FAIL then return FAIL

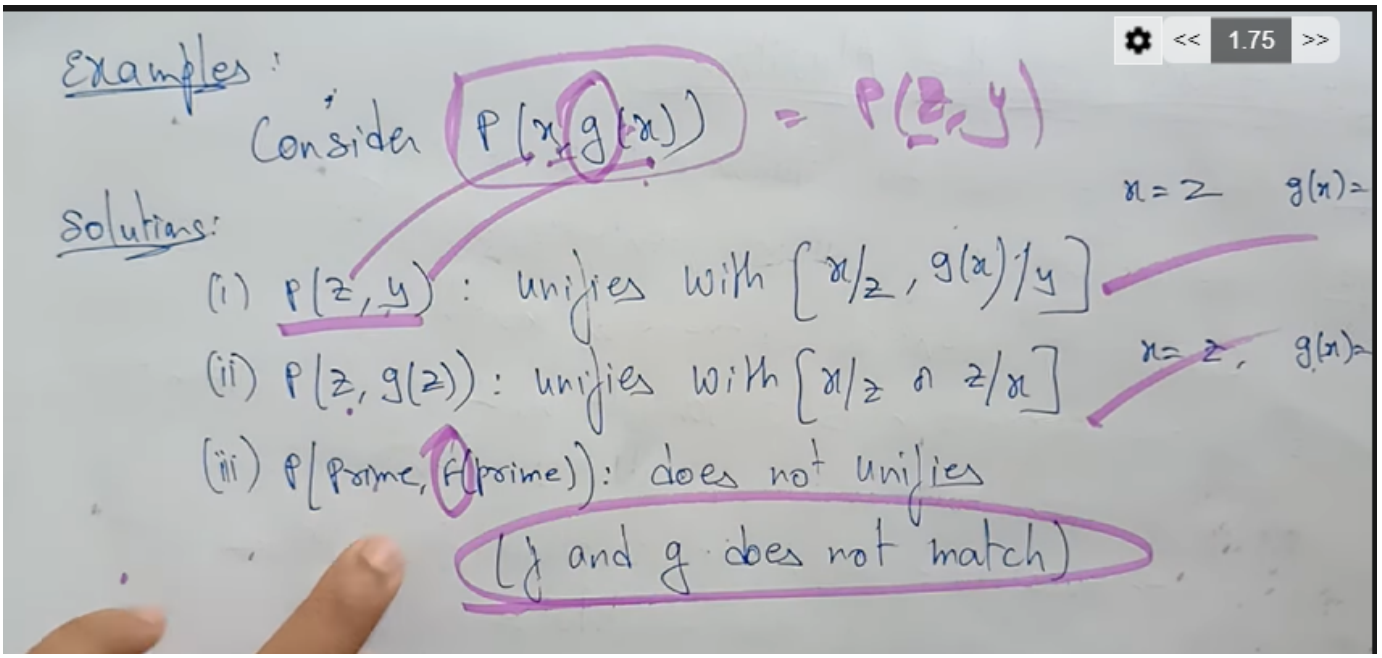
c) If S is not equal to NIL then

Step 6 : Return SUBST


Step3 :- If L1 & L2 have a different
then return FAIL.
Step4 :- Set SUBST to NIL
Step5 :- Loop
Step6 :- Return SUBST
Substitute
{ for i←1 to no: of arguments in L1:
a) Call unify with the ith argument of L1 and the ith argument of L2 putting result in S.
b) If S=FAIL then return FAIL
c) If S is not equal to NIL then
(i) Apply S to the remainder of both L1 & L2
(ii) SUBST = APPEND (S, SUBST) }


Implementation of the Algorithm
Step1 :- Initialize the substitution set to be empty
Step2 :- Recursively unify atomic sentences:
a) check for identical expression match.
b) If one expression is a variable vi, & other is a term ti which does not contain variable vi then:
→ Substitute ti/vi in existing substitutions
→ Add ti/vi to the substitution setlist.
→ If both the expressions are functions, then function name must be similar, & no: of arguments must be same in both expression.


b) If one expression is
which does not contain Variable Vi
→ Substitute ti/Vi in existing
→ Add ti/Vi to the substitution
→ If both the expressions are
must be similar, & no: of argumen
both expression.


1.75
Examples:
Consider P(x, g(x)) = P(z, y)
Solutions:
(i) P(z, y) : unifies with [x/z, g(x)/y]
x = z g(x) =
(ii) P(z, g(z)) : unifies with [x/z or z/x]
x = z, g(x) =
(iii) P(Prime, F(prime)) : does not unifies
(f and g does not match)

Resolution in FOLL

→ Resolution is a theorem proving technique that ~~proceeds~~ ~~by building~~ ~~to~~ proofs by contradictions.

→ It is used, if there are various stmts are given, & need to prove a conclusion of those stmt.

→ Unification is a key concept in proofs by resolutions.

→ Resolution is a single inference rule which can efficiently operate on Conjunctive normal form or clausal form.

clause: Disjunction of literals is called clause.

→ Resolution is a single inference rule which can efficiently operate on Conjunctive normal form or clausal form.

clause: Disjunction of literals is called clause.

Conjunctive NF: A sentence represented as a conjunction of clauses is said to be CNF.

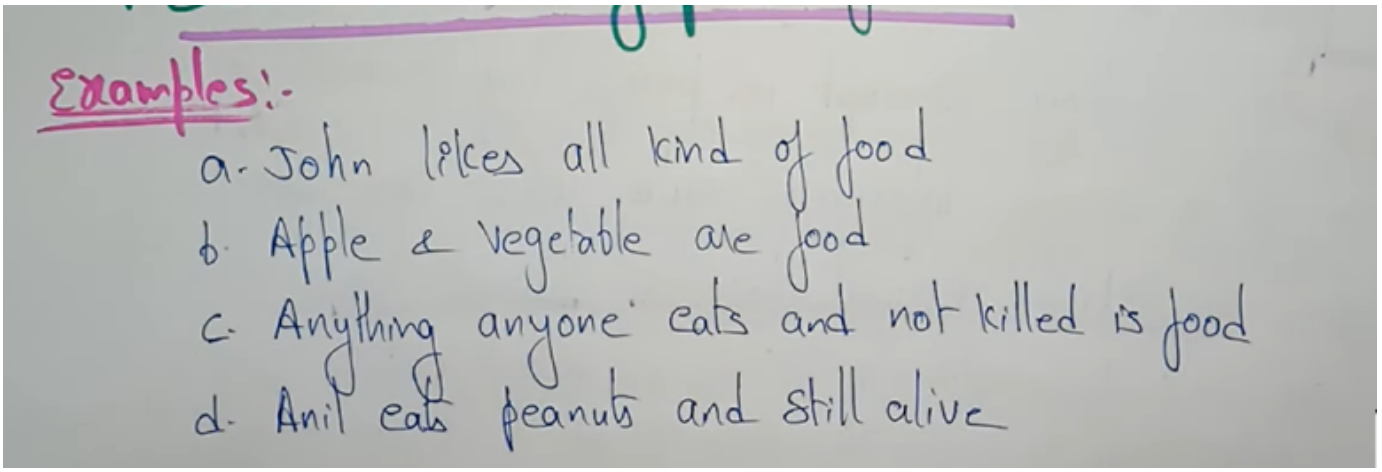
Steps for Resolution.

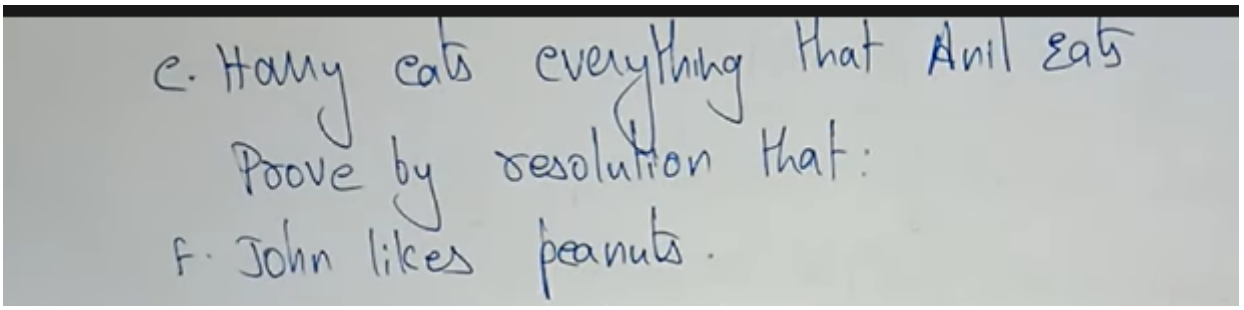
1. Conversion of facts into FOL
2. Convert FOL stmt into CNF
3. Negate the stmt which needs to proove (by contrad
4. Draw resolution graph (unification)

Examples:-

a. John likes all kind of food
b. Apple & vegetable are food
c. Anything anyone eats and not killed is food

Prove resolution





Sol:

Step 1 :- Conversion of facts into FOL

a. $\forall x : food(x) \rightarrow likes(John, x)$

b. $food(Apple) \wedge food(vegetables)$

c. $\forall x \forall y : eats(x, y) \wedge \neg killed(x) \rightarrow food(y)$

d. $eats(Anil, peanuts) \wedge alive(Anil)$

e. $\forall x : eats(Anil, x) \rightarrow eats(Harry, x)$

f. $\forall x : \neg killed(x) \rightarrow alive(x)$ } added predicates

g. $\forall x : alive(x) \rightarrow \neg killed(x)$ }

h. $likes(John, Peanuts)$

Step 2 :- Conversion of FOL into CNF

(b/s CNF makes easier for resolution proofs)

(i) Eliminate all implications ($\rightarrow$) & rewrite

(a) $\forall x \neg food(x) \vee likes(John, x)$

(b) $food(Apple) \wedge food(vegetables)$

(c) $\forall x \forall y \neg [eats(x, y) \wedge \neg killed(x)] \vee food(y)$

(d) $eats(Anil, peatnuts) \wedge alive(Anil)$

(e) $\forall x \neg eats(Anil, x) \vee eats(Harry, x)$

(f) $\forall x \neg [\neg killed(x)] \vee alive(x)$

(g) $\forall x \neg alive(x) \vee \neg killed(x)$

(ii) → Move negation (¬) inwards and rewrite

(a) $\forall x \neg food(x) \vee likes(John, x)$

(b) $food(apple) \wedge food(vegetables)$

(c) $\forall x \forall y \neg eats(x, y) \vee killed(x) \vee food(y)$

(d) $eats(Anil, peanuts) \wedge alive(Anil)$

(e) $\forall x \neg eats(Anil, x) \vee eats(Harry, x)$

(f) $\forall x \neg killed(x) \vee alive(x)$

(g) $\forall x \neg alive(x) \vee \neg killed(x)$

(h) $likes(John, peanuts)$

(iii) Rename Variables or Standardize variables

(a) $\forall x \neg food(x) \vee likes(John, x)$

(b) $food(Apple) \wedge food(vegetables)$

(c) $\forall y \forall z \neg eats(y, z) \vee killed(y) \wedge food(z)$

(d) $eats(Anil, peanuts) \wedge alive(Anil)$

(e) $\forall w \neg eats(Anil, w) \vee eats(Harry, w)$

(f) $\forall g \neg killed(g) \vee alive(g)$

(g) $\forall k \neg alive(k) \vee \neg killed(k)$

(h) $likes(John, peanuts)$

(iv) Eliminate existential instantiation quantifier by elimination

{ but in this prob there are no $\exists$ so all stmts remain s

(v) Drop universal quantifiers:

$\forall$

(i) Drop universal quantifiers:

(a) ¬ food(x) V likes (John, x)
(b) food (Apple)
(c) food (Vegetables)
(d) ¬ eats (y, z) V killed (y) V food (z)
(e) eats (Anil, peanuts)
(f) alive (Anil)
(g) ¬ eats (Anil, w) V eats (Harry, w)
(h) killed (g) V alive (g)
(i) ¬ alive (k) V ¬ killed (k)
(j) likes (John, peanuts)

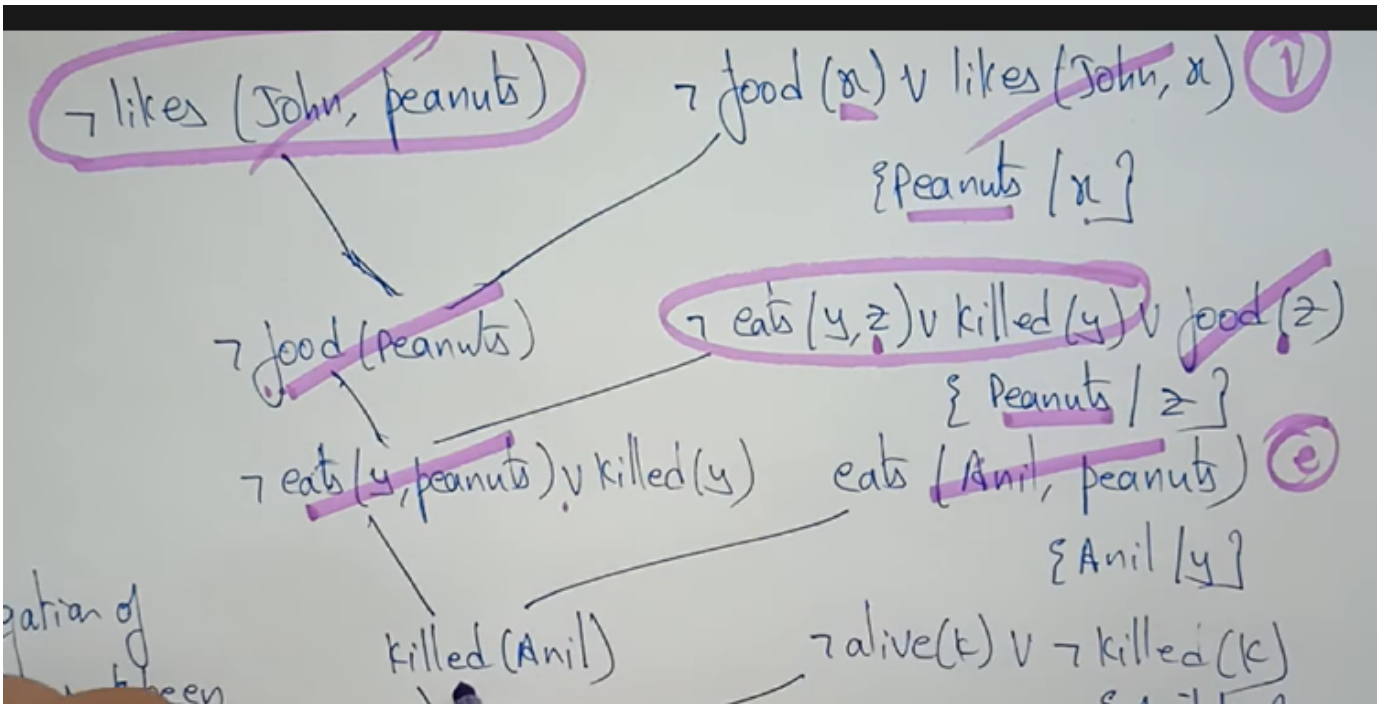
This step will not make ...

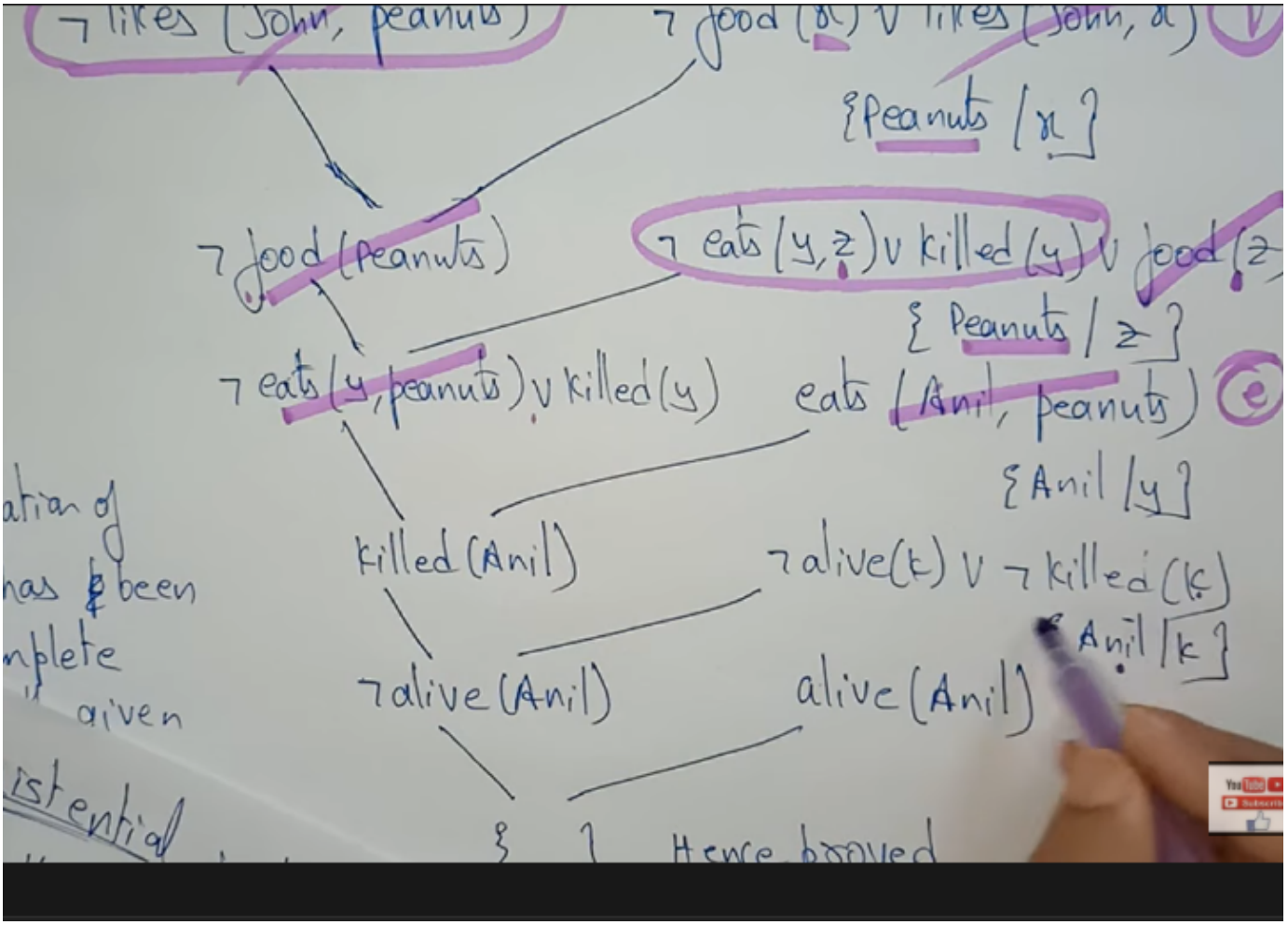
Step 3: Negate the statement to be proved.

In this stmt, we will apply negation to the conclusion stmt, which will be written as ¬ likes (John, peanuts).

Step 4: Draw Resolution graph:

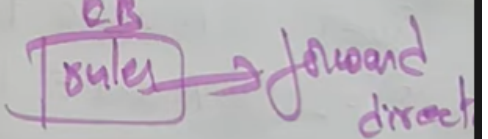
Now, in this step, we will solve the problem by resolution tree using substitution, for the above problem it will be given as:


¬ likes (John, peanuts)
¬ food (x) ∨ likes (John, x)
{Peanuts / x}
¬ food (peanuts)
¬ eats (y, z) ∨ killed (y) ∨ food (z)
{ Peanuts / z }
¬ eats (y, peanuts) ∨ killed (y)
eats (Anil, peanuts)
{Anil / y}
killed (Anil)
¬ alive (k) ∨ ¬ killed (k)


¬ likes (John, peanuts)
{Peanuts / x}
¬ food (peanuts)
¬ eats (y, z) ∨ killed (y)
{ Peanuts / z }
¬ eats (y, peanuts) ∨ killed (y)
eats (Anil, peanuts)
{Anil / y}
killed (Anil)
¬ alive (k) ∨ ¬ killed (k)
{Anil / k}
¬ alive (Anil)
alive (Anil)
{ } Hence proved

Forward chaining:-

⇒ It is a form of reasoning which starts with atomic sentences in the knowledge base and applies inference rules in the forward direction to extract more data until a goal is reached.



Properties :-

→ It moves from bottom to up (top)

→ It is a process of making a conclusion based on known facts of data, by starting from the initial state and reach the

Properties :-

→ It moves from bottom to up (top)

→ It is a process of making a conclusion based on known facts of data, by starting from the initial state and reach the goal st

→ Forward chaining approach is also called as data-driven as

to the goal using available data.

→ Forward - chaining approach is commonly used in the expert System.

Example :

Rule 1 : If A and C then F
Rule 2 : If A and E then G
Rule 3 : If B then E
Rule 4 : If G then D

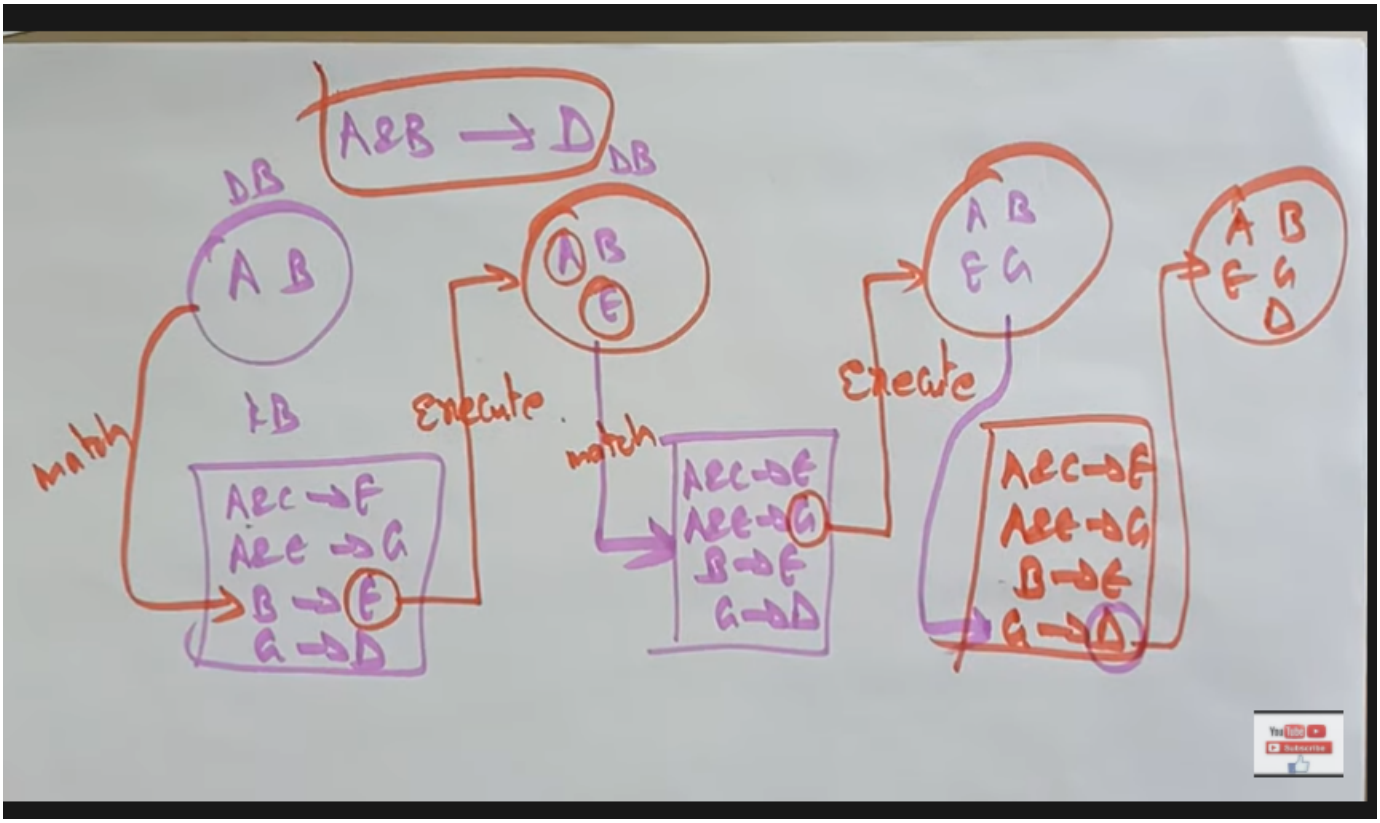
Database

A B

knowledge base

A & C → F
A & E → G
B → E
G → D

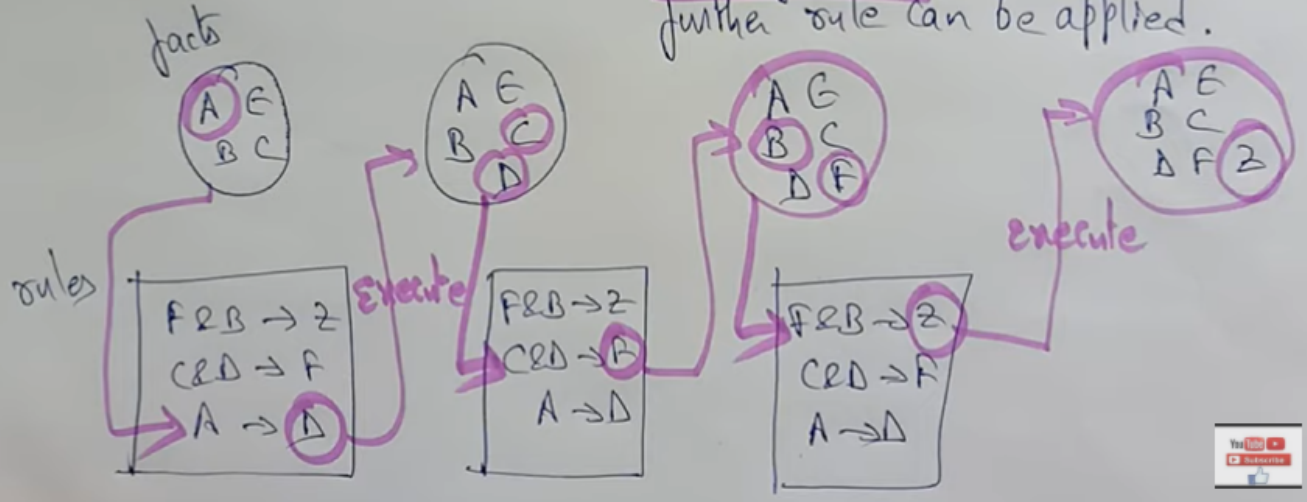
Problem : Proove if A and B true, then D is true.



Example :- Forward chaining

Goal state : Z

termination condition: Stop if Z is derived (or) no further rule can be applied.



Backward Chaining :-

→ A Backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining thro rules to find known facts support the goal.

Properties of backward chaining :-

→ It is known as a top-down approach

→ Backward-chaining is based on modus ponens inference rule.

→ In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.

→ It is called a goal-driven approach, as a list of goal

Properties of backward chaining :-

→ It is known as a top-down approach

→ Backward-chaining is based on modus ponens inference rule.

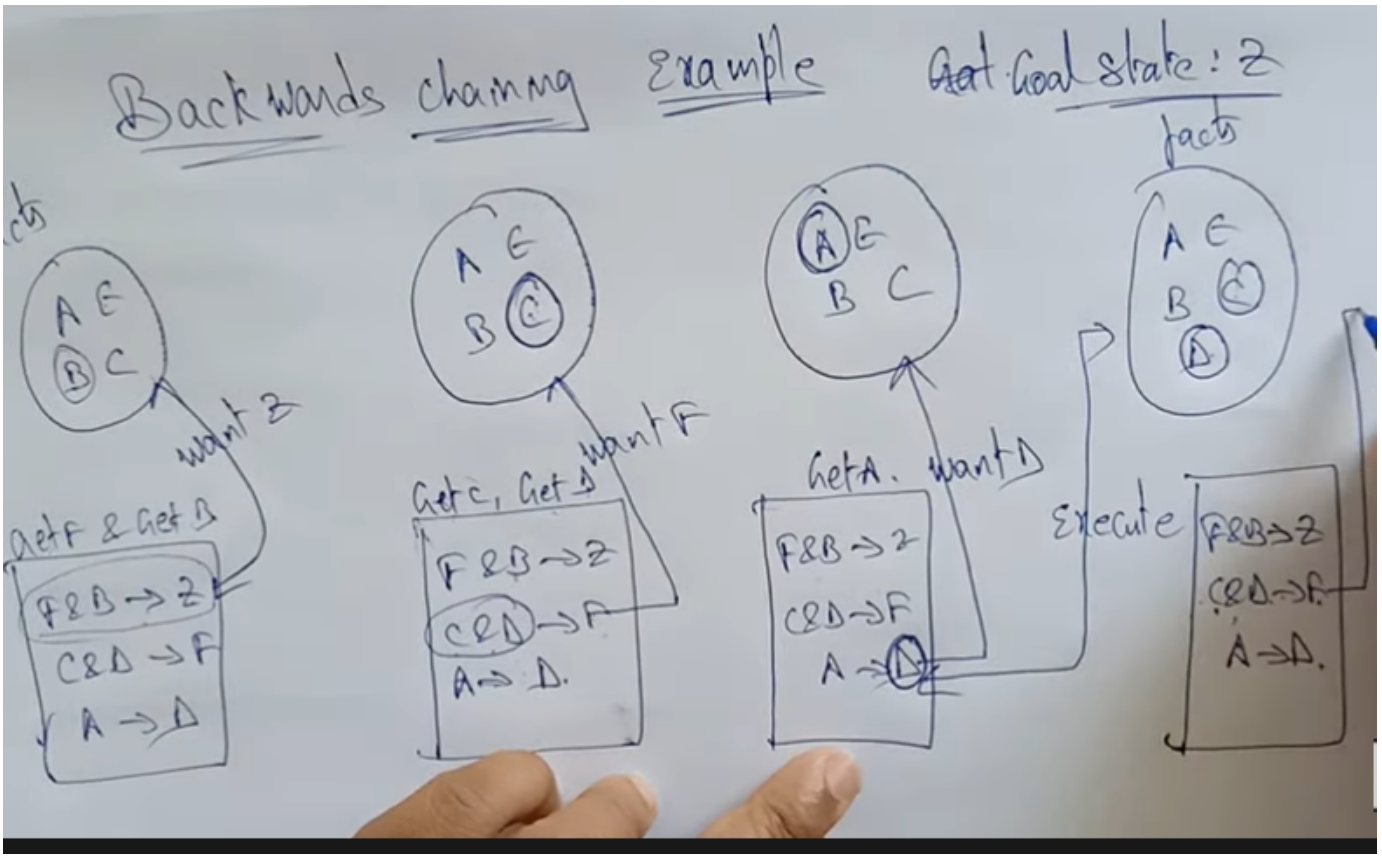
→ In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.

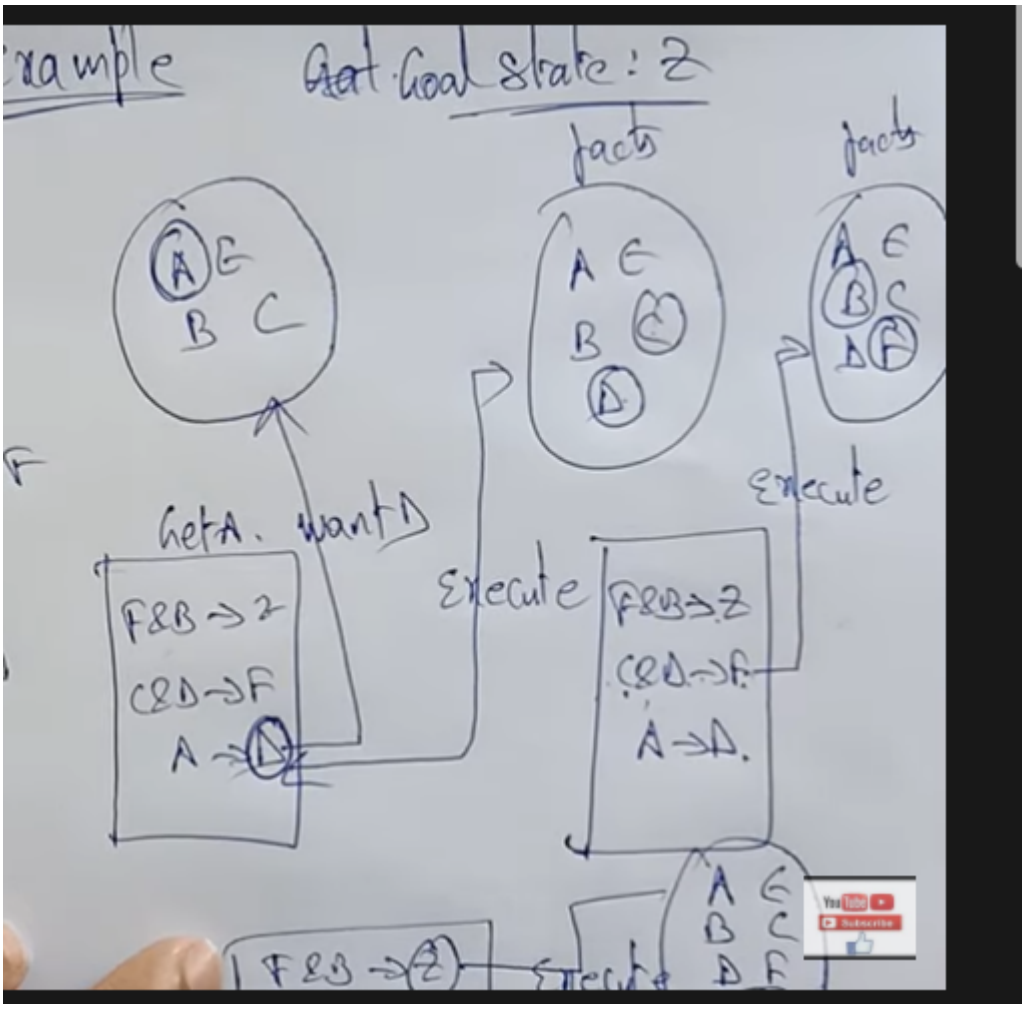
→ It is called a goal-driven approach, as a list of goal decides

Which rules are selected and used

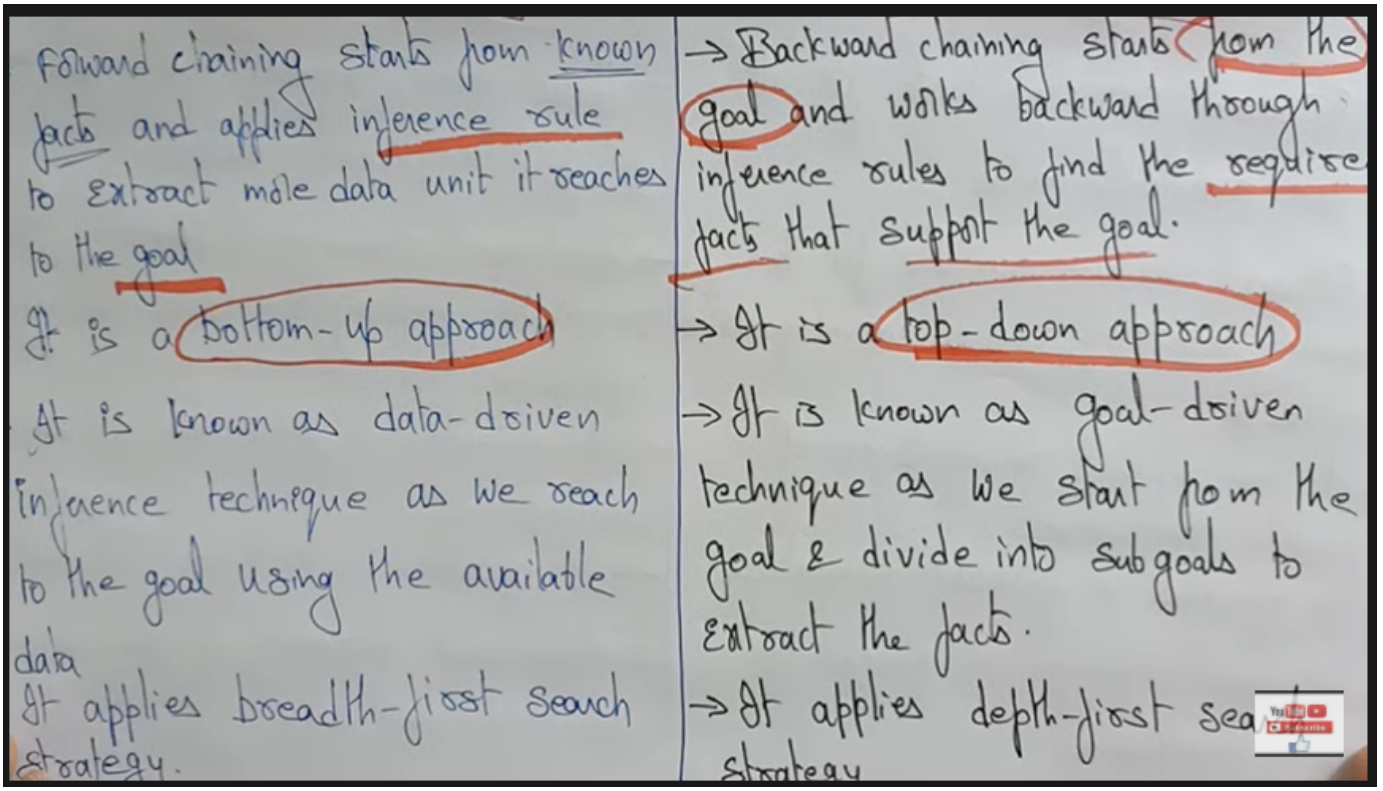
Backward-chaining algorithm is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.

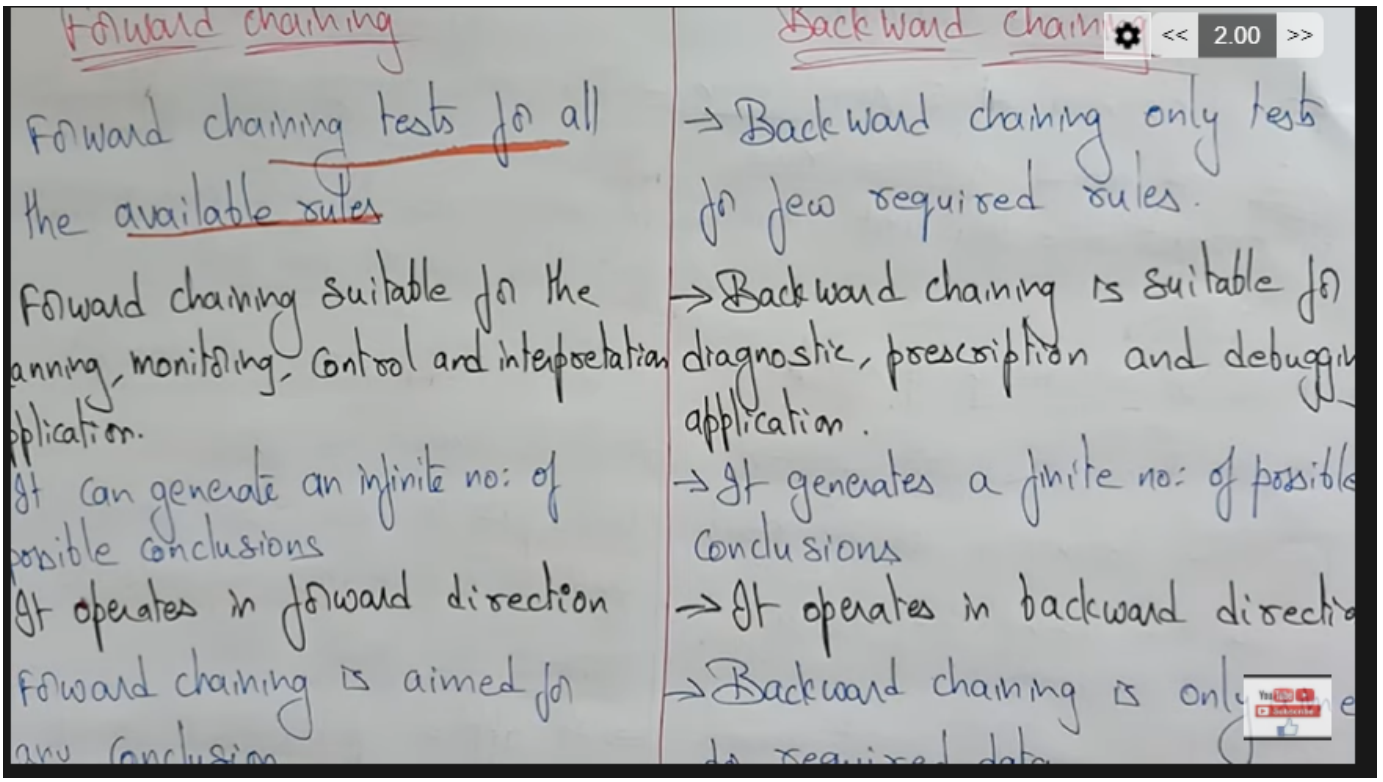
The backward-chaining method mostly used a depth-first search strategy for proof.

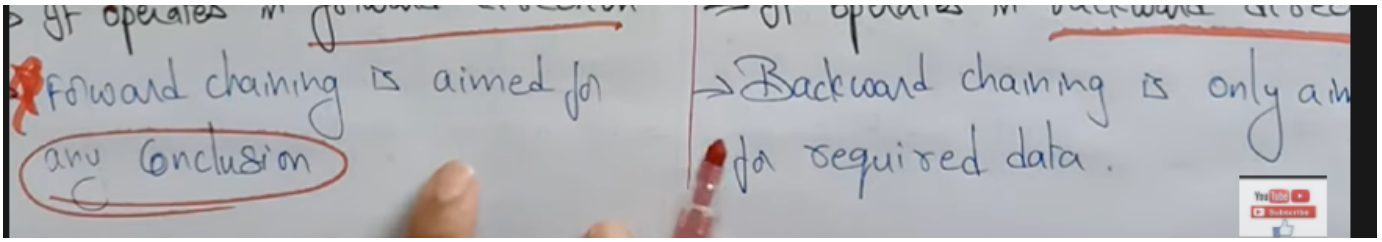

Backwards chaining Example
Goal state : Z
facts
A E
B C
want Z
Get F & Get B
F & B → Z
C & D → F
A → D
A E
B C
Get C, Get D
want F
F & B → Z
C & D → F
A → D.
A E
B C
Get A. want D
F & B → Z
C & D → F
A →
A E
B C
D
Execute
F & B → Z
C & D → F.
A → D.


Goal state : Z
facts
facts
A E
B C
A E
B C
D F
A E
B C
D
Execute
Execute
Get A. want D
F&B → Z
C&D → F
A → D
F&B → Z
C&D → F
A → D.
A E
B C
D F
F&B → Z
Execute

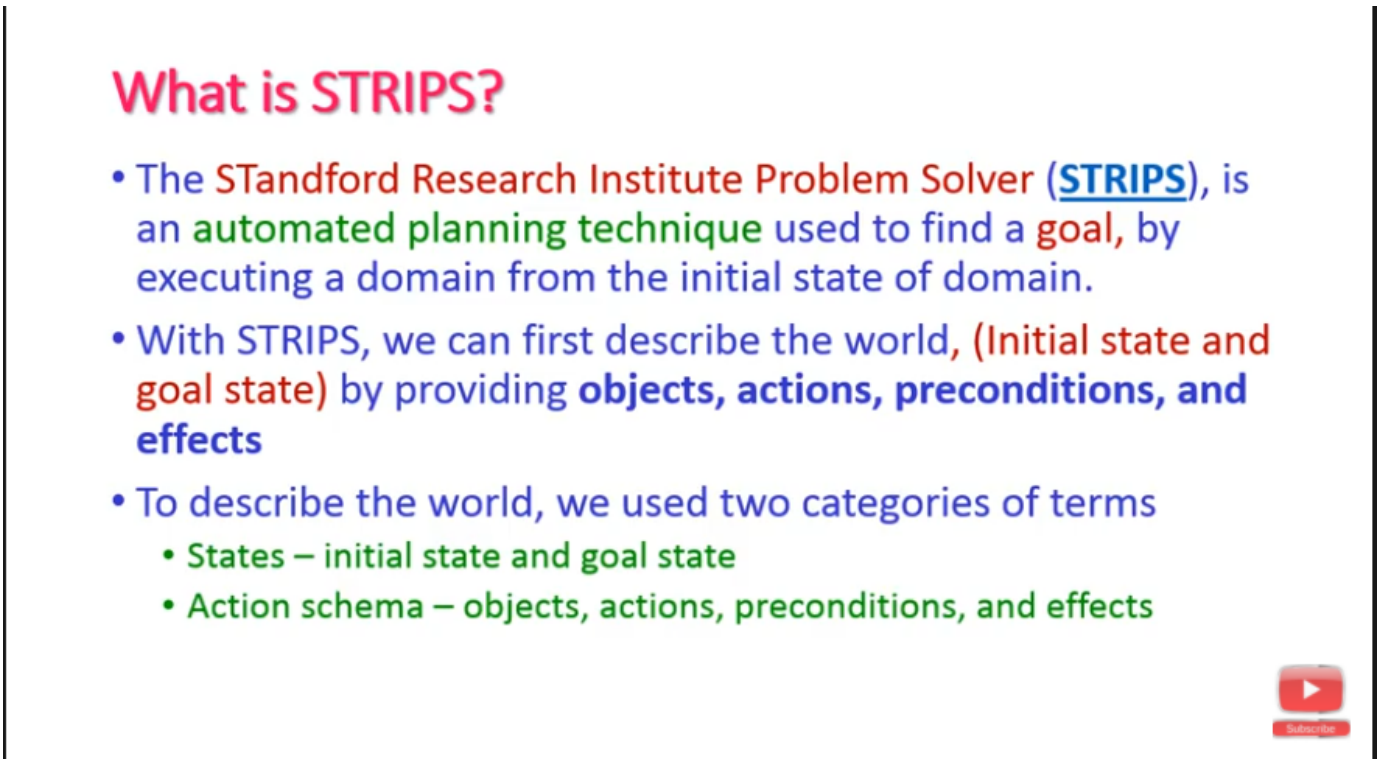
Fc vs Bc


Forward chaining starts from known facts and applies inference rule to extract more data unit it reaches to the goal
It is a bottom-up approach
It is known as data-driven inference technique as we reach to the goal using the available data
It applies breadth-first search strategy.
→ Backward chaining starts from the goal and works backward through inference rules to find the required facts that support the goal.
→ It is a top-down approach
→ It is known as goal-driven technique as we start from the goal & divide into subgoals to extract the facts.
→ It applies depth-first sea
strategy


Forward chaining
Forward chaining tests for all the available rules
Forward chaining suitable for the anning, monitoring, control and interpretation pplication.
It can generate an infinite no: of possible conclusions
It operates in forward direction
Forward chaining is aimed for any conclusion
Backward chain
→ Backward chaining only tests for few required rules.
→ Backward chaining is suitable for diagnostic, prescription and debuggi application.
→ It generates a finite no: of possible conclusions
→ It operates in backward direct
→ Backward chaining is onl
<< 2.00 >>



Planning problem in AI



- Once the world is described, then provide a problem set.
- A problem consists of an initial state and a goal condition.
- STRIPS can then search all possible states, starting from the initial one, executing various actions, until it reaches the goal.

Unit4

Planning Domain Definition Language (PDDL).

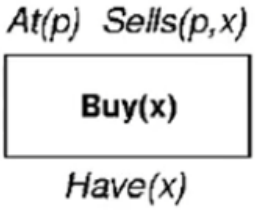
- A common language for writing STRIPS domain and problem sets, is the Planning Domain Definition Language (**PDDL**).
- In PDDL most of the codes are English words, so that it can be clearly read and well understood.
- It's a relatively easy approach to writing simple AI planning problems.

STRIPS - States, Goals and Actions

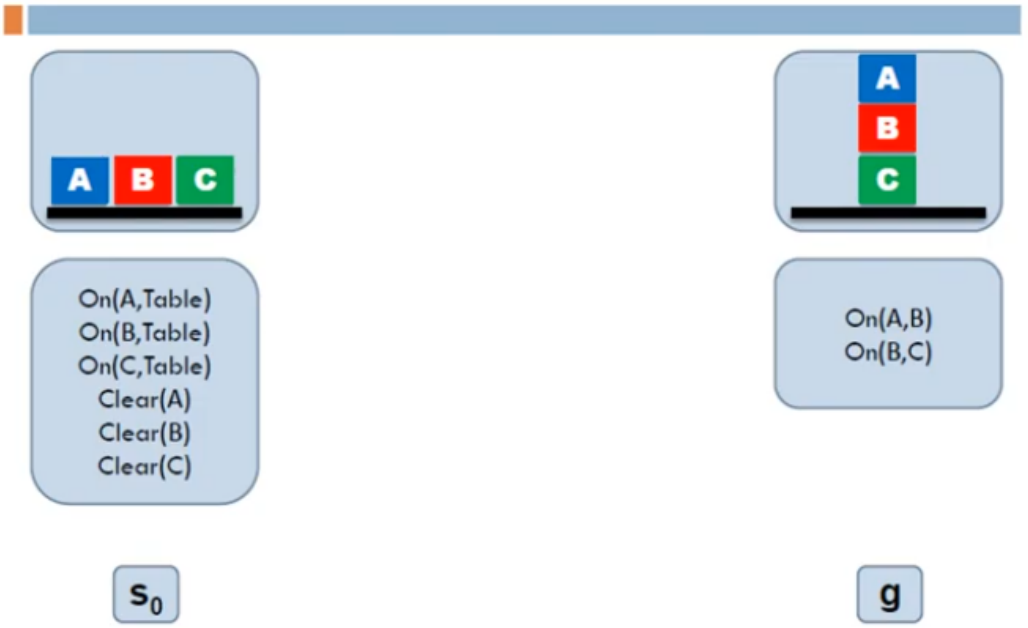
- States: conjunctions of ground, function-free, and positive literals, such as At(Home) ^ Have(Banana)
- To describe states, Closed-world assumption is used (the world model contains everything, the agent needs to know: there can be no surprise)
- Goals: conjunctions of literals, may contain variables (existential), goal may represent more than one state
  - E.g. At(Home) ^ ¬Have(Bananas)
  - E.g. At(x) ^ Sells(x, Bananas)
- Actions: preconditions that must hold before execution and the effects after execution

STRIPS Action Schema

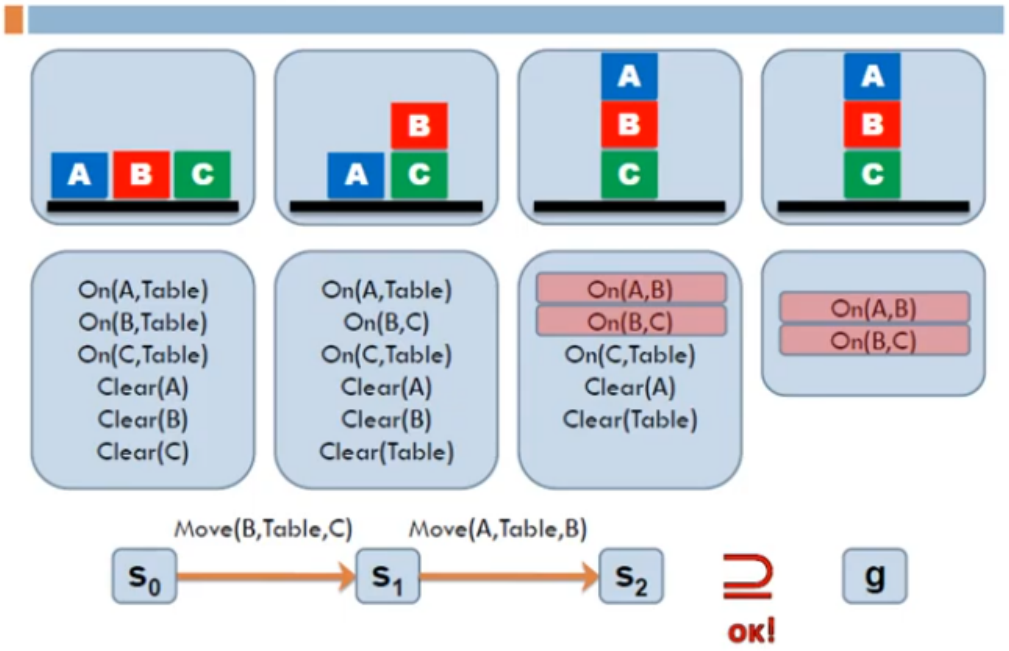
- An action schema includes:
  - Action name & parameter list (variables)
  - Precondition: a conjunction of function-free positive literals. The action variables must also appear in precondition.
  - Effect: a conjunction of function-free literals (positive or negative)
- Add-list: positive literals
- Delete-list: negative literals
- Example:
  - Action: Buy (x)
  - Precondition: At (p), Sells (p, x)
  - Effect: Have(x)



STRIPS planning



STRIPS planning



Air Cargo Transport

- An air cargo transport problem involving loading cargo onto planes, and unloading cargo off of planes, and flying the planes from one place to another place.
- The problem can be defined with three actions:
  - Load, Unload, and Fly.
- The actions affect two predicates:
  - In(c, p) means that cargo c is inside plane p, and At (x, a) means that object x (either plane or cargo) is at airport a.
- The following plan is a solution to the problem:

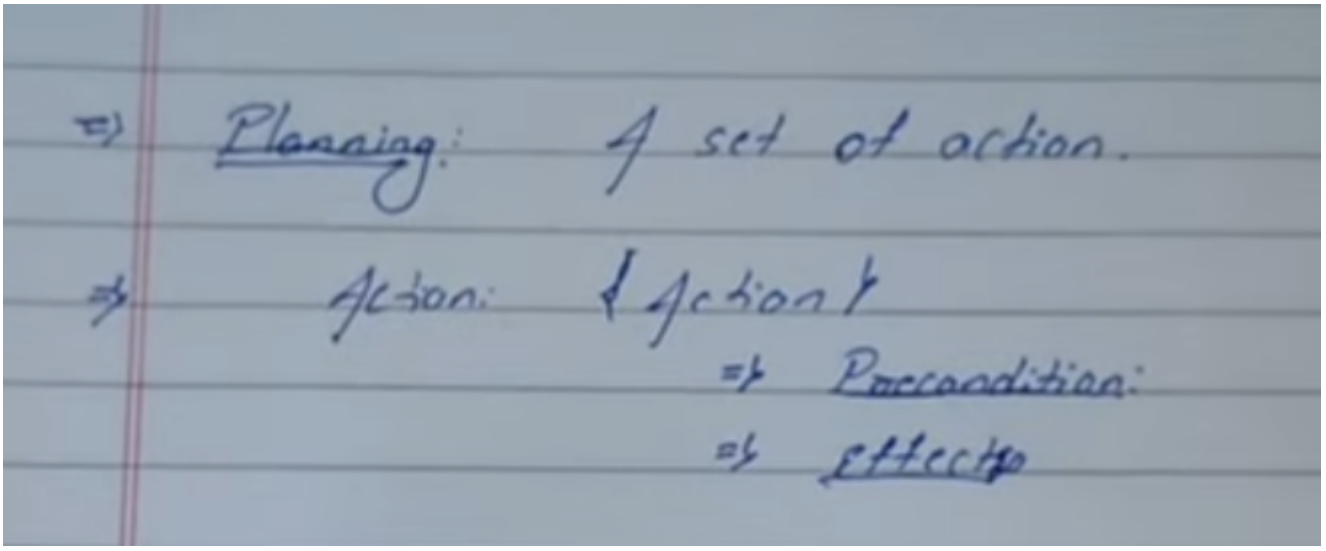
  [Load(C1, P1, SFO), Fly(P1, SFO, JFK)

  Load(C2, P2, JFK), Fly(P2, JFK, SFO)]

A STRIPS problem involving transportation of air cargo between airports.

$Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK)$
$\land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2)$
$\land Airport(JFK) \land Airport(SFO))$
$Goal(At(C_1, JFK) \land At(C_2, SFO))$
$Action(Load(c, p, a),$
PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $\neg At(c, a) \land In(c, p))$
$Action(Unload(c, p, a),$
PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $At(c, a) \land \neg In(c, p))$
$Action(Fly(p, from, to),$
PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
EFFECT: $\neg At(p, from) \land At(p, to))$

Refer ppt for clear explanation of both examples



Q. Spare tire problem:-

- Flat tire → Axle.
- Spare → Trunk.

→ Goal state: Spare :→ Axle.
Flat → Trunk.

→ STRIPS:

1) Remove Spare
2) Remove Flat
3) Put Spare → Axle
4) Put Flat → Trunk.

-> 1) Action (Remove (spare, trunk))
- precond: At(spare, trunk).
- effects ¬ At (trunk, spare).

2) Action (Remove (Flat, axle))
- Precond: At (flat, axle)
- Effect: ¬ At (flat, axle).

3) Action (Put (flat, trunk))
- Precond: ¬ At(flat, axle).
- Effect: At (flat, trunk).

4) Action (Put (spare, axle)).
- Precond: ¬ At (spare, trunk).
- effect: At (spare, axle).

Planning with State-Space Search

- The most straightforward approach of planning algorithm, is state-space search
  - Forward state-space search (Progression)
  - Backward state-space search (Regression)
- The descriptions of actions in a planning problem, and specify both preconditions and effects
- It is possible to search in both direction: either forward from the initial state or backward from the goal
- We can also use the explicit action and goal representations, to derive effective heuristics automatically.

Problem Formulation for Progression

- Initial state:
  - Initial state of the planning problem
- Actions:
  - Applicable to the current state.
  - First actions' preconditions are satisfied, Successor states are generated
  - Add positive literals to add list and negative literals to delete list.
- Goal test:
  - Whether the state satisfies the goal of the planning
- Step cost:
  - Each action is 1 (assumed)

Progression

- From initial state, search forward by selecting operators whose preconditions can be unified with literals in the state
- New state includes positive literals of effect; the negated literals of effect are deleted
- Search forward until goal unifies with resulting state
- This is forward state-space search using STRIPS operators

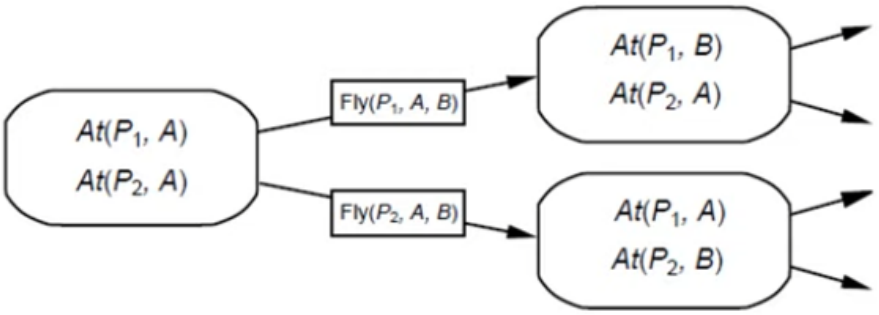
Eg:

Example : Transportation of air cargo between airports.

$$
\begin{aligned}
&Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \\
&\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \\
&\quad \land Airport(JFK) \land Airport(SFO)) \\
&Goal(At(C_1, JFK) \land At(C_2, SFO)) \\
&Action(Load(c, p, a), \\
&\quad \text{PRECOND}: At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT}: \neg At(c, a) \land In(c, p)) \\
&Action(Unload(c, p, a), \\
&\quad \text{PRECOND}: In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT}: At(c, a) \land \neg In(c, p)) \\
&Action(Fly(p, from, to), \\
&\quad \text{PRECOND}: At(p, from) \land Plane(p) \land Airport(from) \land Airport(to) \\
&\quad \text{EFFECT}: \neg At(p, from) \land At(p, to))
\end{aligned}
$$

Forward (progression) state-space search

- Starting from the initial state and using the problem's actions to search forward for the goal state.



FSSS Algorithm

- (1) compute whether or not a state is a goal state,
- (2) find the set of all actions that are applicable to a state, and
- (3) compute a successor state, that is the result of applying an action to a state
- Algorithm takes as input the statement P = (O, s0, g) of a planning problem P. (O contains a list of actions)
- If P is solvable, then Forward-search(O, s0, g) returns a solution plan; otherwise it returns failure.

Algorithm for FSSS

1. Forward-search(O, s0, g)
2. s ← s0
3. π ← the empty plan
4. loop
   1. if s satisfies g then return π
   2. applicable ← {a | a is a ground instance of an operator in O, and precond(a) is true in s}
   3. if applicable = ∅ then return failure
   4. Non-deterministically choose an action a ∈ applicable
   5. s ← γ(s, a)
   6. π ← π.a

o->list of operations
S0 initial state

Problem Formulation for Regression

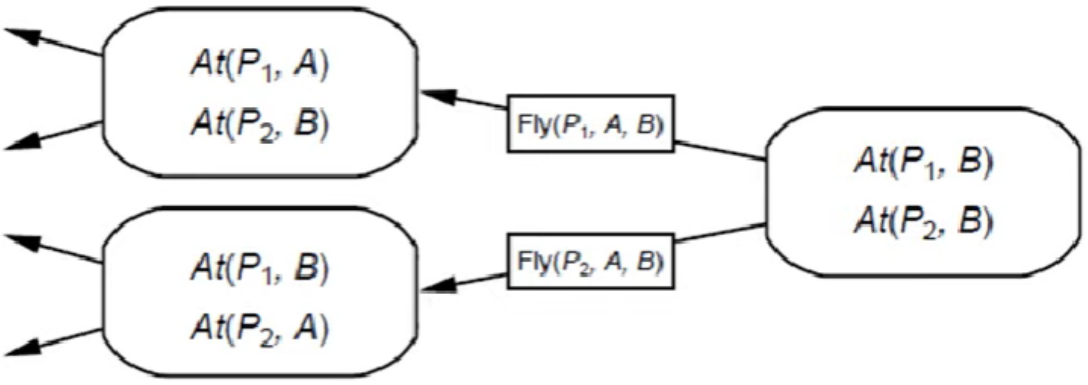
- Initial state:
  - Initial state of the planning problem
- Actions:
  - Applicable to the current state.
  - First actions' preconditions are satisfied, Successor states are generated
  - Add positive literals to add list and negative literals to delete list.
- Goal test:
  - Whether the state satisfies the goal of the planning
- Step cost:
  - Each action is **1** (assumed)

Regression

- The goal state must unify with at least one of the positive literals in the operator's effect
- Its preconditions must hold in the previous situation, and these become subgoals which might be satisfied by the initial conditions
- Perform backward chaining from goal
- Again, this is just state-space search using STRIPS operators

Backward (regression) state-space search:

- Backward state search starting from the goal state(s)
- Using the inverse of the actions to search backward for the initial state.



BSSS Algorithm

- Start at the goal,
- Test the goal is initial state, otherwise
- Apply inverses of the planning operators to produce subgoals,
- The algorithm will stop, if we produce a set of subgoals that satisfies the initial state.

BSSS Algorithm

1. Backward-search(O, $s0$, g)
2. π the empty plan
3. loop
   1. if $s0$ satisfies g then return π
   2. $applicable \leftarrow \{a \mid a$ is a ground instance of an operator in O that is relevant for $g\}$
   3. if $applicable = \emptyset$ then return failure
   4. Non-deterministically choose an action $a \in applicable$
   5. $\pi \leftarrow a.\ \pi$
   6. $g \leftarrow \gamma^{-1}(g, a)$

Heuristics for State-Space Search

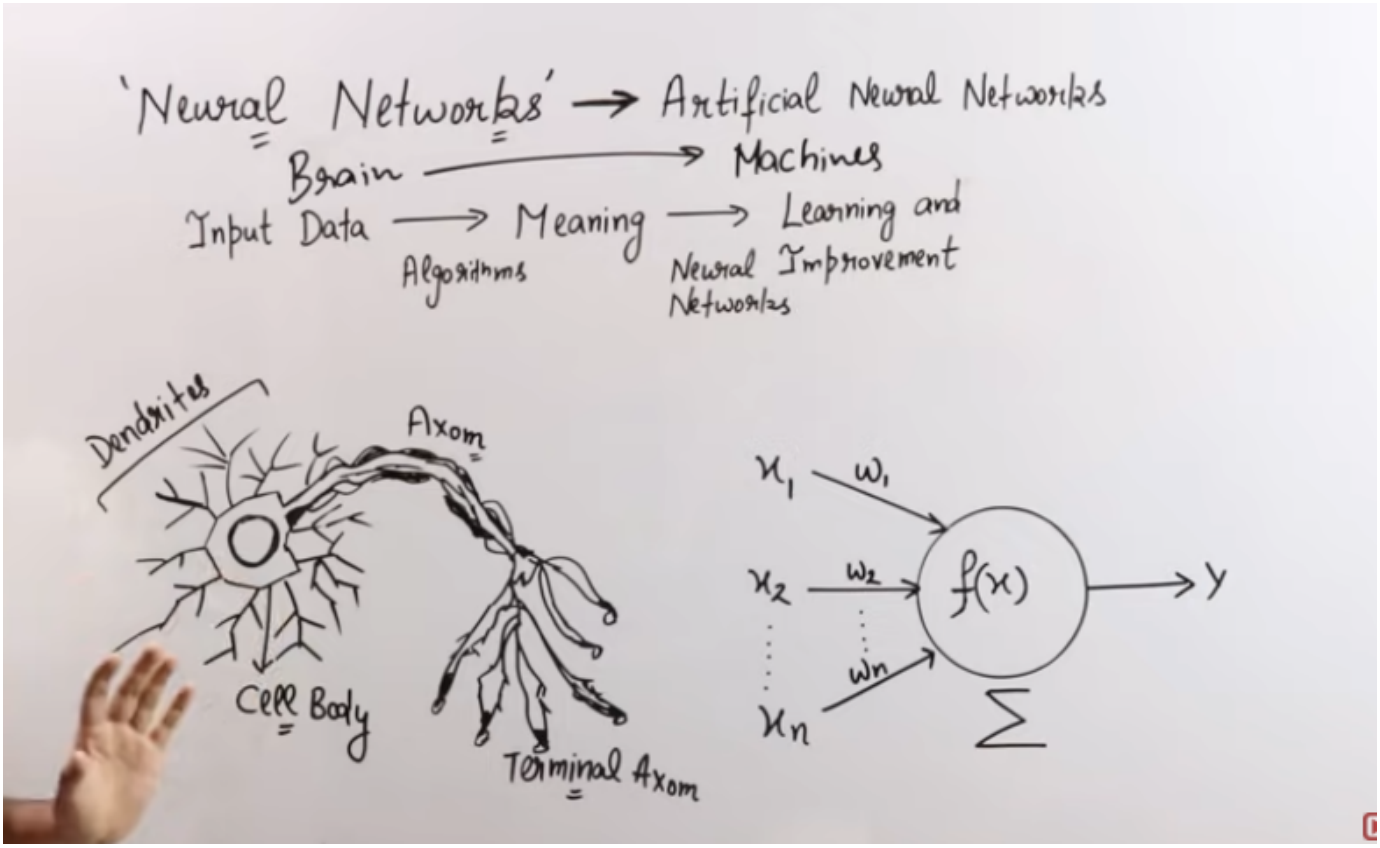
- How to find an admissible heuristic estimate?
  - Distance from a state to the goal?
  - Look at the effects of the actions and at the goals and guess how many actions are needed
- NP-hard
- Relaxed problem
- Subgoal independence assumption:
- The cost of solving a conjunction of subgoals, is approximated by the sum of the costs of solving each subgoal independently

Relaxation Problem

- Idea: removing all preconditions from the actions

- Which implies, the number of steps required to solve a conjunction of goals, and the number of unsatisfied goals
  - There may be two actions,
  - Some actions deletes the goal literal achieved by the other actions
  - some action may achieve multiple goals
- Combining relaxation with subgoal $\rightarrow$ exact # of unsatisfied goals

Part2: ann


'Neural Networks' → Artificial Neural Networks
Brain → Machines
Input Data → Meaning → Learning and Improvement
Algorithms
Neural Networks
Dendrites
Axon
Cell Body
Terminal Axon
x_1 w_1
x_2 w_2
w_n
x_n
$f(x)$
y
Σ

Neural Networks → Artificial Neural Networks

Brain → Machines

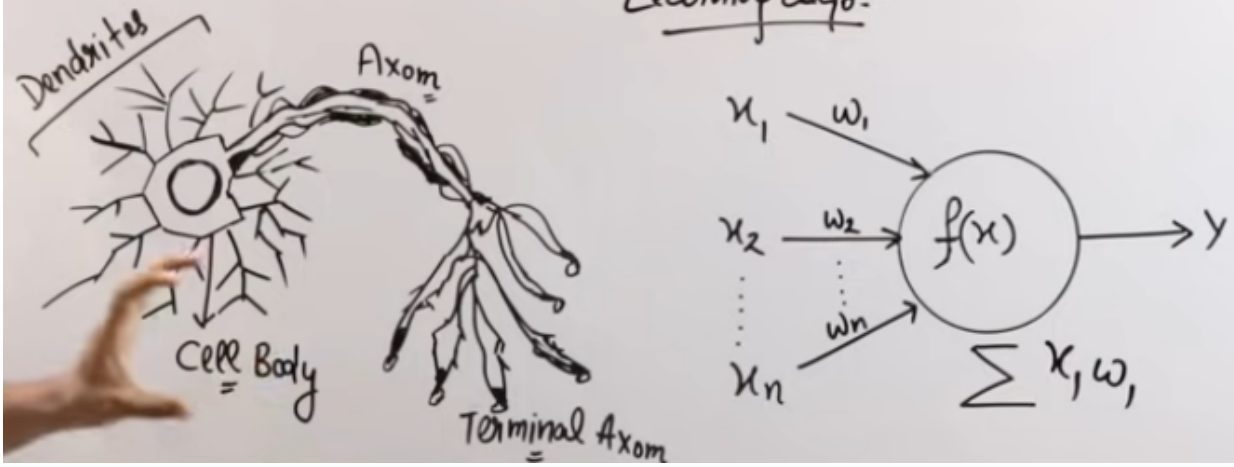
Input Data → Meaning → Learning and Improvement

Algorithms

Neural Networks

8^8

Learning algo.

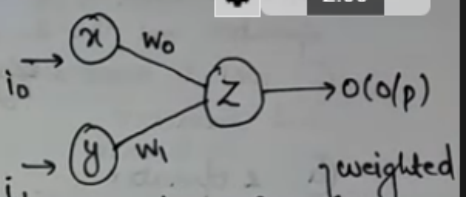


(A133) Easy Engineering Classes – Free YouTube Lectures

EEC Classes GGSIPU, UPTU, Mumbai Univ., Pune Univ., GTU, Anna Univ., PTU and Others EEC Classes

<< 2.00 >>

Ques:- Draw and explain architecture of Artificial Neural N/W. what is output and input for Artificial Neural N/W? what are the advantages of Neural N/W over Conventional Computers?



(ANN) is an info^n processing paradigm that is inspired by the way of biological nervous system.

→ Configured for specific appl^n → i) Data classification, ii) Pattern Recognition

→ MODEL → i) Interconnections, ii) Activation Function, iii) Learning Rules.

Characteristics of ANN:

→ i) Neurally Implemented mathematical model.

ii) Huge no. of interconnected processing elements called neurons for processing.

iii) i/p signals arrive at processing elements through Conn^n and Connected weights.

Advantages:

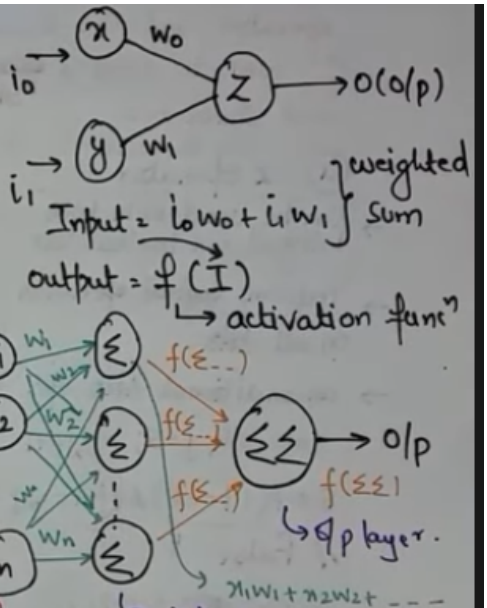
→ i)

ii)

...

Ques:- Draw and explain architecture of Artificial Neural N/W. what is output and input for Artificial Neural N/W? what are the advantages of Neural N/W over Conventional Computers?

(ANN) is an info processing paradigm that is inspired by the way of biological nervous system.
- Configured for specific appln → ii) Data classification, iii) Pattern Recognition
- MODEL → i) Interconnections, ii) Activation Function, iii) Learning Rules.



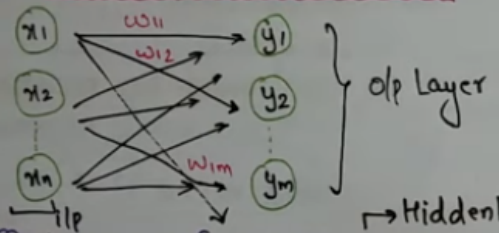
Characteristics of ANN:
- i) Neurally implemented mathematical model.
- ii) Huge no. of interconnected processing elements called neurons for processing.
- iii) i/p signals arrive at processing elements through Conn^n and Connected weights.

Advantages:
- i) Ability to learn and model non-linear and complex relationship.
- ii) Easy Generalization.
- iii) No Restriction on i/p Variable.

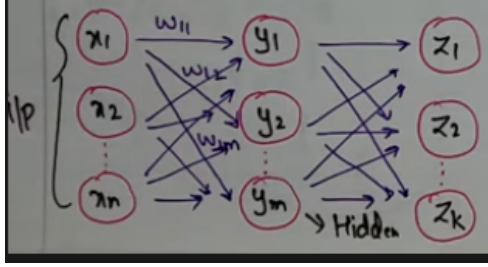
(AI-17) Easy Engineering Classes – Free YouTube Lectures
EEC Classes GGSIPU, UPTU, Mumbai Univ., Pune Univ., GTU, Anna Univ., PTU and Others EEC Classes

Ques:- Explain types of Artificial Neural Network architectures.

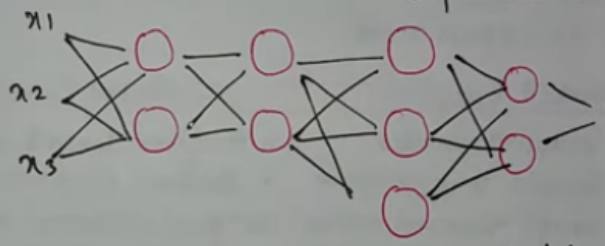
① SINGLE LAYER FEED FORWARD N/W: → only two layers → i/p, o/p



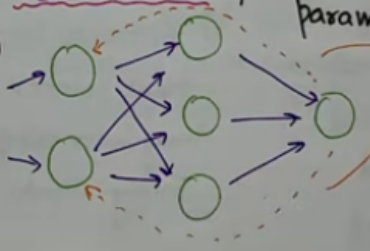
② MULTI LAYER FEED FORWARD N/W: → Hidden Layer ↓ Computationaly more Stronger.



③ MULTILAYER PERCEPTRON: → 3 or more layers are used to classify non-linearly Separable data.



④ FEEDBACK ANN: feedback is provided to adjust parameters.



Ques:- Explain MCP Neuron Model. [linear-threshold Gate Model].

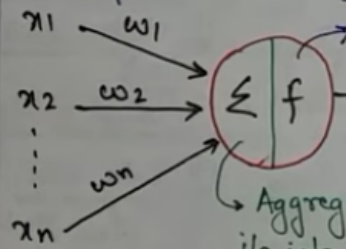
McCulloch Pitts Neuron model. [in 1943 by Warren McCulloch and Walter Pitts]

↳ First mathematical model of biological neuron.

↳ Basic building block of neural n/w.

↳ Directed Weight Graph is used for Connecting neurons.

↳ Two possible States of neuron → Active (1), Silent (0)



→ produces the o/p using threshold (T)

$o/p = \begin{cases} 1 & \text{if } X > T \\ 0 & \text{if } X < T \end{cases}$

Aggregates the weighted i/p into Singe numeric Value.

$\Sigma x_1w_1 + x_2w_2 + \dots + x_nw_n$ (X)

BIAS/ Threshold:- Minimum Value of weighted active i/p for a neuron to fire.

if effective i/p is larger than T, -then o/p → (1) Else (0).

o/p:- f(a) ↳ $\Sigma w_i x_i - T$

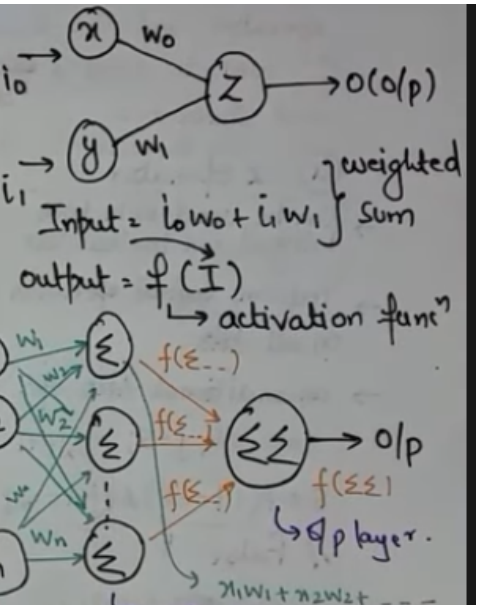
fun^n] $Q(n) = 1$ if $n > 0$, 0

Ques:- Draw and explain architecture of Artificial Neural N/w. what is output and input for Artificial Neural N/w? what are the advantages of Neural N/w over Conventional Computers?

(ANN) is an info^n processing paradigm that is inspired by the way of biological nervous system.

↳ Configured for specific appl^n → i) Data classification, ii) Pattern Recognition

↳ MODEL → i) Interconnections, ii) Activation Function, iii) Learning Rules.



Characteristics of ANN:

↳ i) Neurally Implemented mathematical model.

ii) Huge no. of interconnected processing elements called neurons for processing.

iii) i/p signals arrive at processing elements through Conn^n and Connected weights.

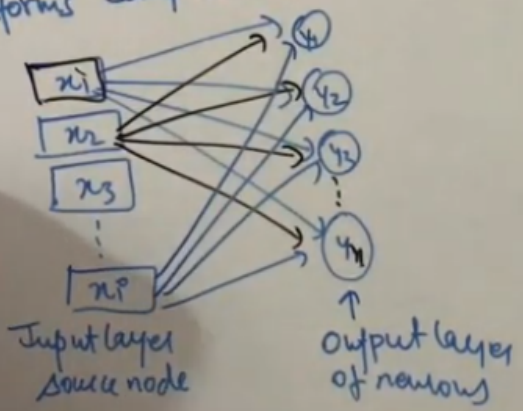
Advantages:

↳ i) Ability to learn and model non-linear and complex relationship.

ii) Easy Generalization.

iii) No Restriction on i/p Variable.

⇒ Despite the two layers, the n/w since it is termed single layer since it is output layer, alone which performs computation.



Single layer feed forward N/w

⇒ this type of network comprises of two layers, namely the input layer and output layer

⇒ Input layer neurons receives the input signals and output layer neurons receives the output signal

⇒ The synaptic links carrying the weights connect every input neurons to output neurons but not vice-versa

⇒ Such type of n/w is called feed forward in type or acyclic in nature

⇒ weight I/h Input-hidden weight
O/h Ouput-hidden weight

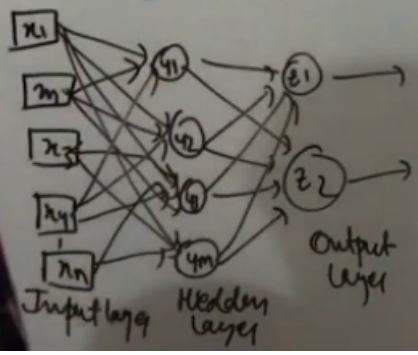
Multi layer feed forward n/w

⇒ A multilayer feedforward network with L input neurons, m_1 neurons in first hidden layer, m_2 neurons in second hidden layer and n output can be written as
$L - m_1 - m_2 - n$

⇒ This second class of feed forward n/w distinguishes itself by presence of one more hidden layer, whose computational nodes are correspondingly called hidden neurons or hidden units

⇒ Hidden layer neurons are present b/w input layer and output layer

⇒ Hidden layer helps in performing useful intermediary computations before directing the input to o/p layer



(A158)

Ques:- What is the importance of activation func^n in ANN? Explain diff. available activation func^n in detail.

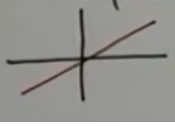
Activation func^n is an internal state of neuron. ↳ used to convert the i/p signal on node of ANN to an o/p signal. ↳ They introduce non-linear properties to N/W.

Weighted Sum of i/p becomes i/p signal to A.F to give one o/p signal.

Without Activation Function: o/p signal would be Simple linear func^n.] Easy to solve but limited in processing complex learning. } Images, video, audio] not work properly.
↳ polynomial of one degree.

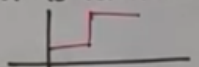
Types:-

(i) **Identity Func^n:-** $f(x) = x$ for all x.
↳ linear func^n, o/p is same as i/p.



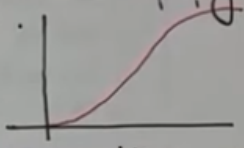
(ii) **Binary Step Func^n:-** used in Single Layer N/W to convert net i/p to o/p [0 / 1]

$$f(x) = \begin{cases} 1 & \text{if } x \geq t \\ 0 & \text{if } x < t \end{cases}$$ t → threshold Value.



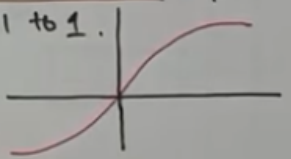
(iii) **Sigmoidal Func^n:-** used in Backpropagation N/W. Range is 0 to 1.

$$f(x) = \frac{1}{1+\exp^{(-x)}} = \frac{1}{1+e^{-x}}$$



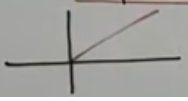
(iv) **Hyperbolic Tangent Func^n (Tanh):** optimization is easy. Range is -1 to 1.

$$f(x) = \frac{1-e^{2x}}{1+e^{-2x}}$$



(v) **Ramp Func^n:-**

$$R(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$



(A143)

Ques:- Discuss Supervised, unsupervised and reinforced learning in neural n/w.

Supervised Learning: Happens in presence of Supervisor.
↳ o/p is already known.
↳ machine is fed with lots of i/p data. [i/p ↔ o/p] } Labelled Dataset.
↳ Model is built to predict the outcomes.
↳ Fast Learning mechanism with high accuracy.
↳ Includes Regression & Classification problems
↳ SVM
↳ KNN
↳ ANN
↳ Decision Trees.

x_1	x_2	y
1	1	1
1	0	1
0	1	1
0	0	?

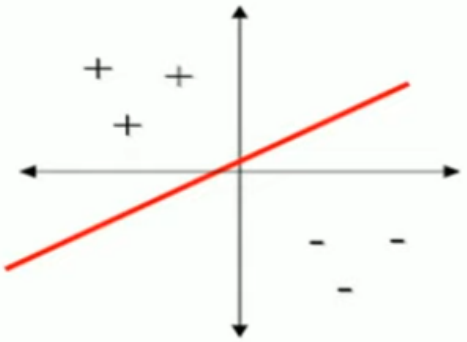
↳ i/p → o/p. (0 0) → o/p ?

unsupervised Learning:- happens without help of Supervisor.
↳ Independent learning process.
↳ no o/p mapping with i/p.
↳ Unlabelled dataset.
↳ Detect Hidden Patterns
↳ clustering and Association Rule mining → Apriori.
K-means

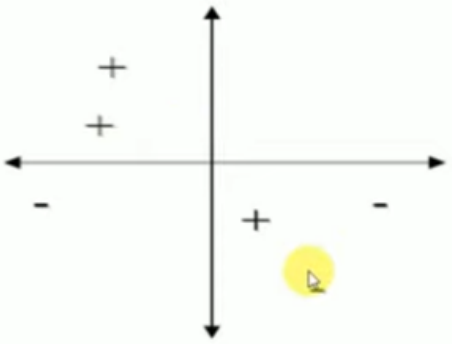
Reinforcement Learning:- Learns from Feedback and past experiences.
↳ long term iterative process.
↳ more feedback ⇒ More Accurate System
↳ Also called **Markov Decision Process.**
↳ Eg:- Robots Training, Self-Driving Car
↳ Deep Adversial N/W, Q-Learni

Supervised Learning Algorithm	Unsupervised Learning Algorithm
i) o/p is known for every i/p	Unknown.
ii) Labelled dataset is used for Learning.	finds hidden Patterns or Association among data items.
iii) Predictive in Nature	Descriptive in nature
iv) Classification, Regression	Clustering, Association Algo
v) Linear Reg, NB, KNN, SVM ...	K-means, Apriori
vi) More Accurate	less.

Gradient Descent and the Delta Rule



Linearly separable

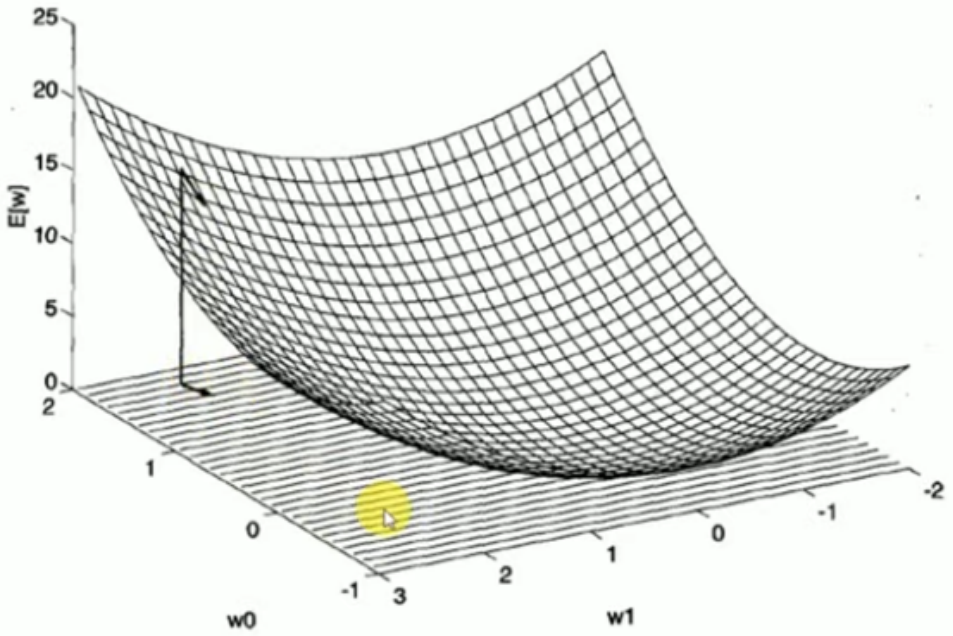


Non-linearly separable

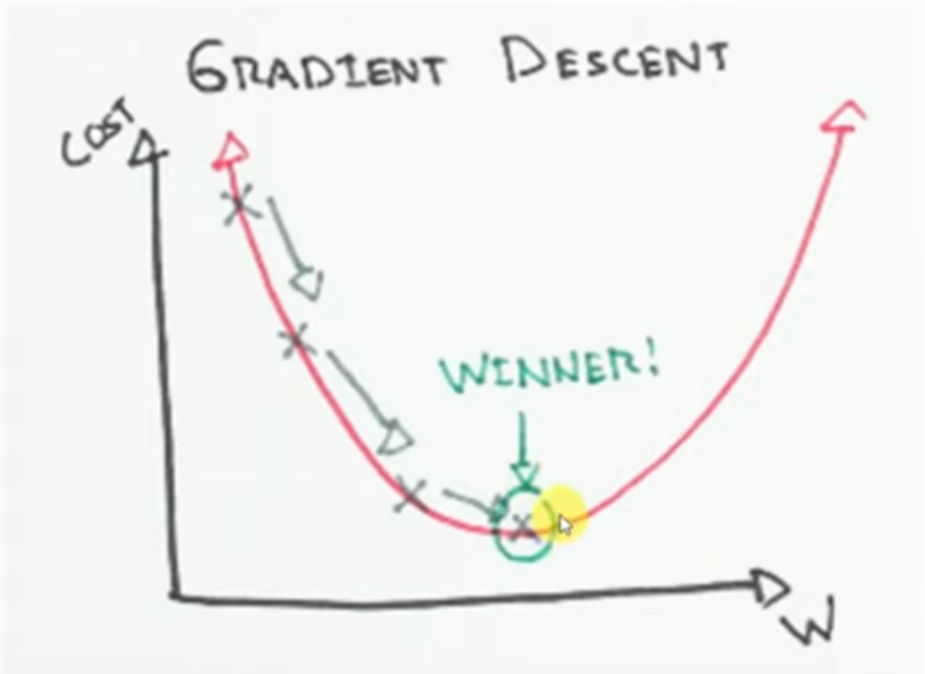
Gradient Descent and the Delta Rule

- Perceptron rule finds a successful weight vector when the training examples are linearly separable, but it can fail to converge if the examples are not linearly separable.
- A second training rule, called the ***delta rule,*** is designed to overcome this difficulty.
- If the training examples are not linearly separable, the delta rule converges toward a best-fit approximation to the target concept.
- The key idea behind the delta rule is to use ***gradient descent*** to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.
- This rule is important because gradient descent provides the basis for the BACKPROPAGATON algorithm, which can learn networks with many interconnected units.
- It is also important because gradient descent can serve as the basis for learning algorithms that must search through hypothesis spaces containing many different types of continuously parameterized hypotheses.

Gradient Descent and the Delta Rule



Gradient Descent and the Delta Rule



Gradient Descent and the Delta Rule

- The delta training rule is best understood by considering the task of training an ***unthresholded*** perceptron; that is, a ***linear unit*** for which the output ***o*** is given by

$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

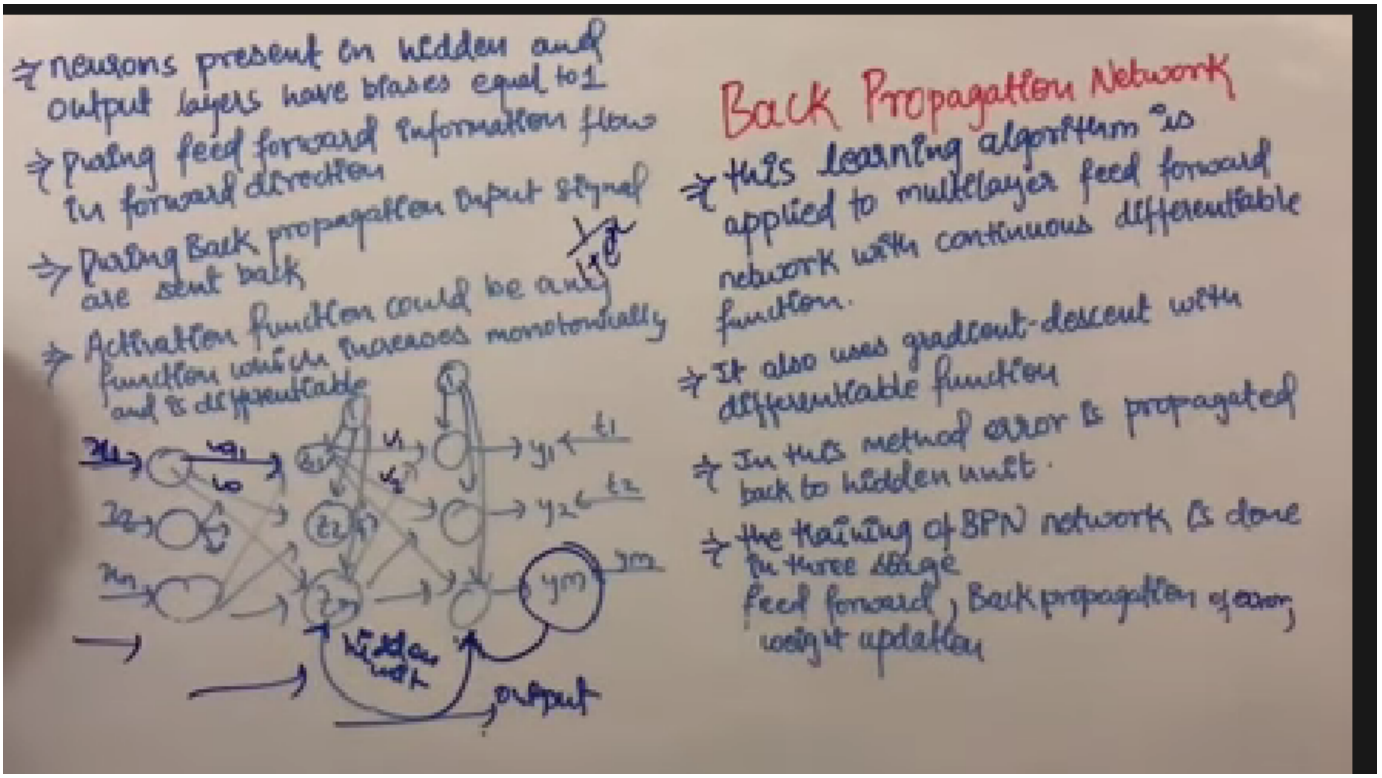
$$o(\vec{x}) = \vec{w} \cdot \vec{x}$$

- Thus, a linear unit corresponds to the first stage of a perceptron, without the threshold.
- In order to derive a weight learning rule for linear units, let us begin by specifying a measure for the ***training error*** of a hypothesis (weight vector), relative to the training examples.
- Although there are many ways to define this error, one common measure is

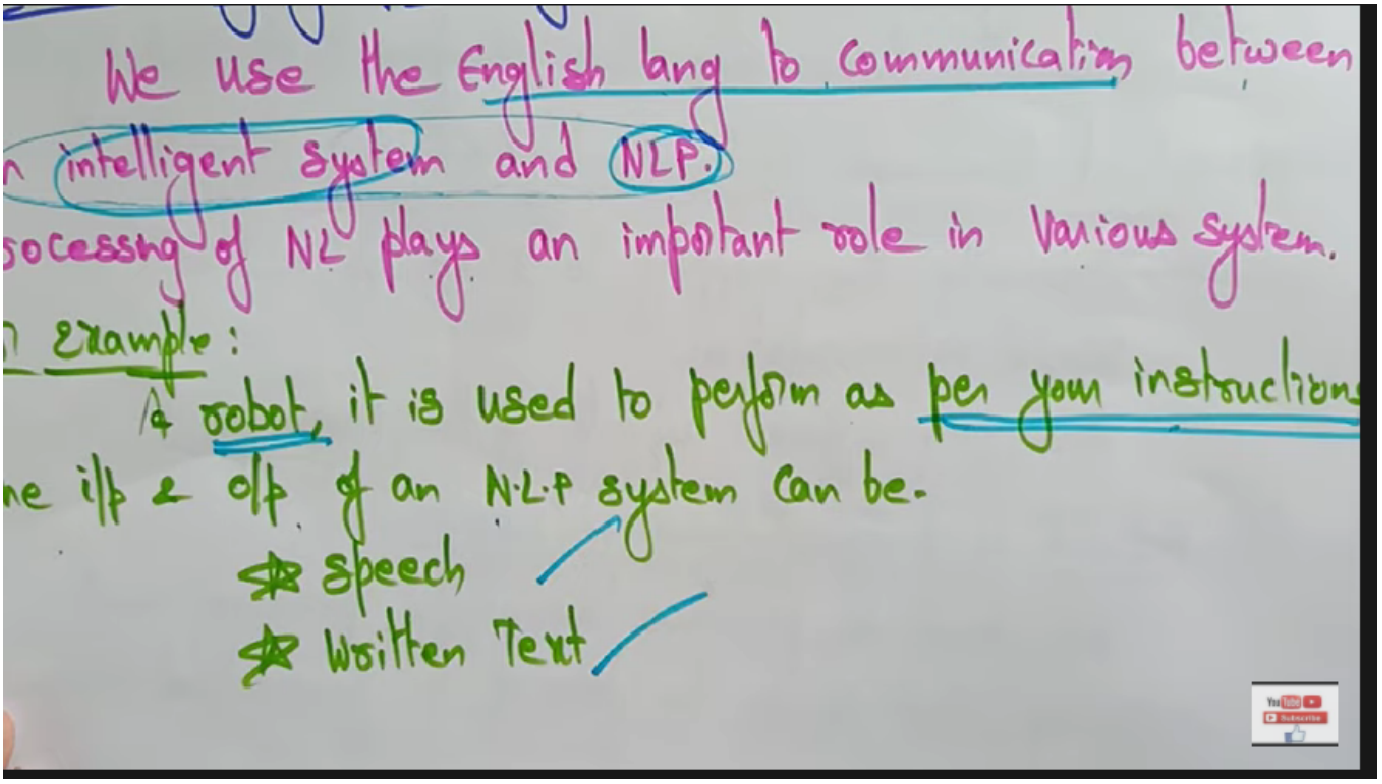
$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

- where D is the set of training examples, ***td*** is the target output for training example *d*, and ***od*** is the output of the linear unit for training example d.

Inkaa undi(malli chudali)



NLP


We use the English lang to communication between
intelligent system and NLP.
processing of NL plays an important role in various system.
example:
robot, it is used to perform as per your instructions
the i/p & o/p of an N.L.P system can be.
speech
Written Text

Components of NLP

Basically there are two components of NLP sys:

(a) Natural language understanding (NLU):

- Basically, the mapping to given i/p in N.L into useful representation.
- Analyzing different aspects of the language.

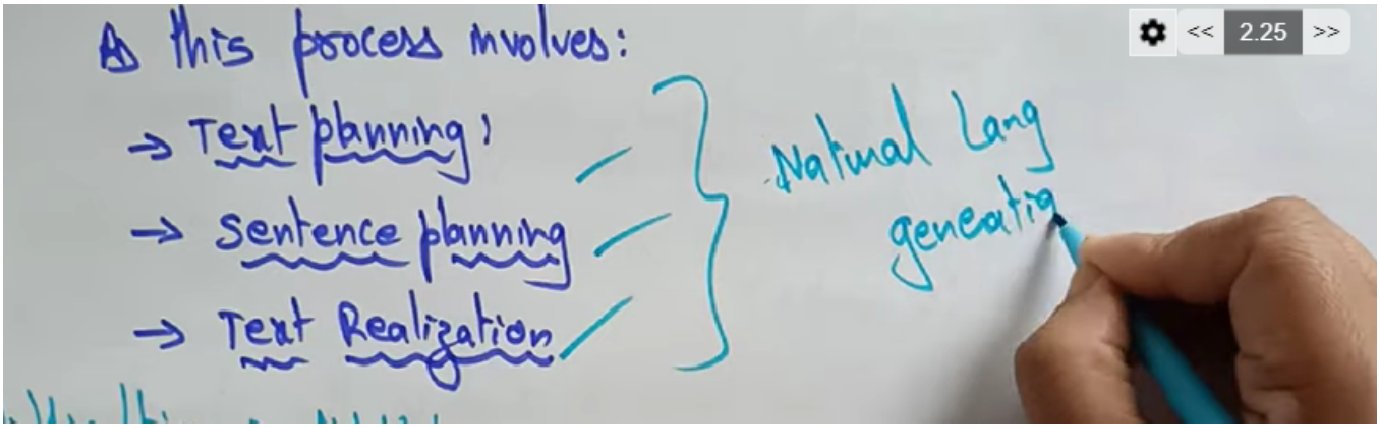
(b) Natural language Generation (NLG):

We have to produce meaningful phrases and

(b) Natural language Generation (NLG):

We have to produce meaningful phrases and sentences. That is in the form of NL from internal

Theater mode (t)



Difficulties in NLU :

(a) Lexical ambiguity :

It's predefined at a very primitive level such as word-level

(b) Syntax level ambiguity

In this, we can define a sentence in a parsed way in a different

(c) Referential ambiguity :-

Referential ambiguity says that we have to ref something using pronouns

teps in NLP : (5 steps)

→ Lexical Analysis : we have to analyze the structure of words. The collection of words and phrases in lang lexicon of a lang

→ Syntactic Analysis (Parsing) :- We use parsing for the analy of the word. Although, have to arrange words in a partic manner. That shows the relationship b/w words

→ Semantic Analysis :

It describes a dictionary meaning which is mean
In the task domain, mapping syntactic structure and objec

→ Discourse Integration :

In this step, the meaning of any sentence depe
upon the meaning of the previous sentence. In addition,
brings the meaning to immediately succeeding sentence.

→ Pragmatic Analysis :

In this step, data is interpreted on what is actua
meant. Although we have to derive aspects of lang which req
real-world knowledge

Natural language Processing (NLP)

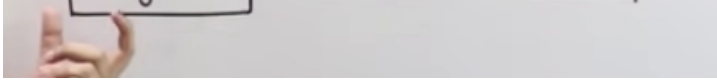
→ How Human Communicate with each other
→ Computer should replicate the same thing
→ Applications of NLP
* Speech Recognition → Siri, Alexa, Assistant
* Sentimental Analysis
* Machine Translations
* Chat bots etc.

NLU ⇒ What do the users say ?
their intent ? Meaning ?

Challenges:
- Lexical Ambiguity
- Syntactic Ambiguity
- Semantic "
- Pragmatic "

→ The tank was full of water.
→ Old men and women were taken to safe place.
→ The Car hit the pole while it was moving.
→ The police are coming.

NLG → What should we say to User ?
→ It should be Intelligent and Conversational.
→ Deal with structured data.
→ Text / Sentence Planning



UNIT 5

NATURAL LANGUAGE PROCESSING

LANGUAGE MODELS:

- Language Models (LMs) estimate the relative likelihood of different phrases and are useful in many different NLP applications.
- For example: They have been used in Twitter Bots for 'robot' accounts to form their own sentences.

Defining Language Models:

- The goal of probabilistic language modelling is to calculate the probability of a sentence of sequence of words:

$$P(W) = P(w_1, w_2, w_3, \ldots, w_n)$$

- And can be used to find the probability of the next word in the sequence:

$$P(w_5 | w_1, w_2, w_3, w_4)$$

NATURAL LANGUAGE PROCESSING

- Initial Method for Calculating Probabilities:

Conditional Probability:

Let A and B be two events with $P(B) \neq 0$. the conditional probability of A given B is:

$$P(A|B) = \frac{P(A,B)}{P(B)}$$

- Chain Rule:

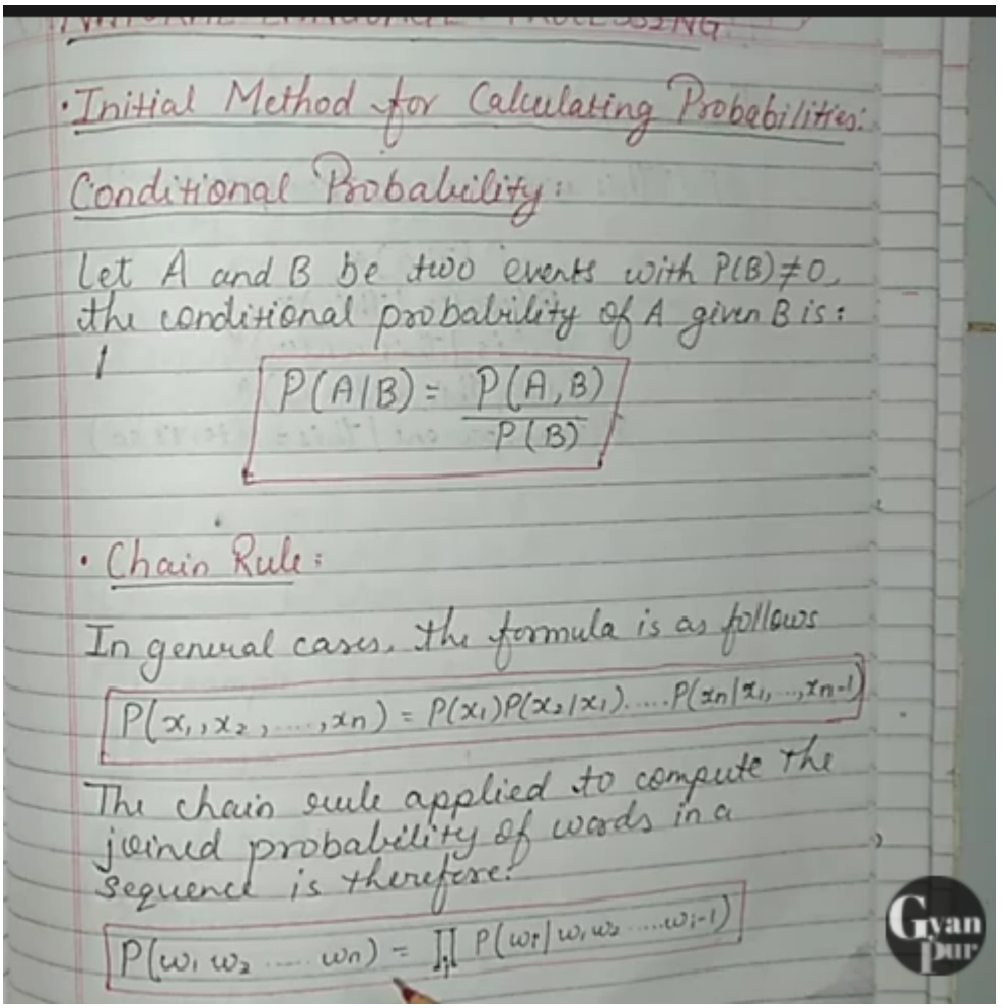
In general cases, the formula is as follo

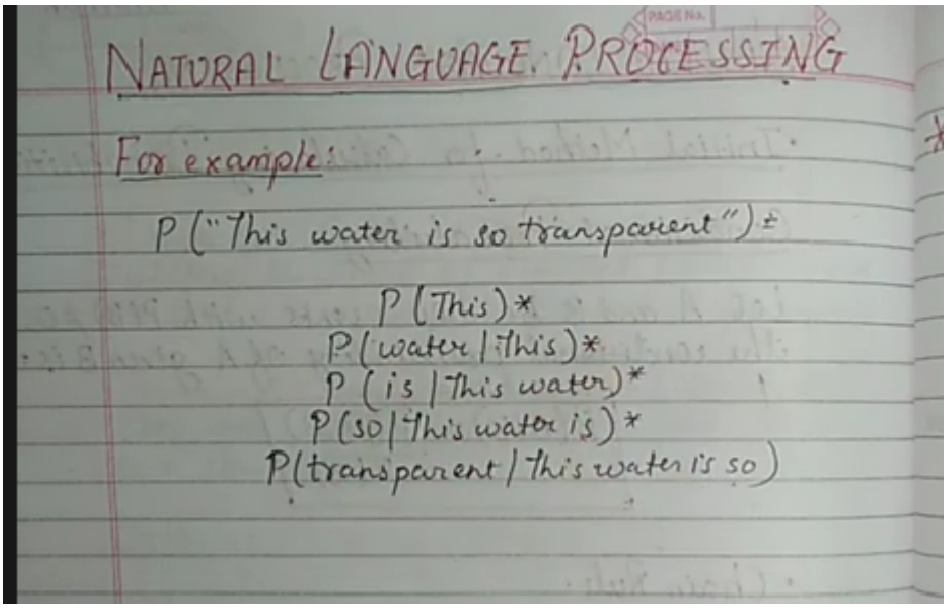
$$P(x_1, x_2, \ldots, x_n) = P(x_1)P(x_2|x_1)\ldots P(x_n|x_1,\ldots$$

The chain rule applied to compute the joined probability of words in a sequence is therefore:

$$P(\quad) = \prod P(w_i | w_1, w_2 \ldots w_{i-1})$$

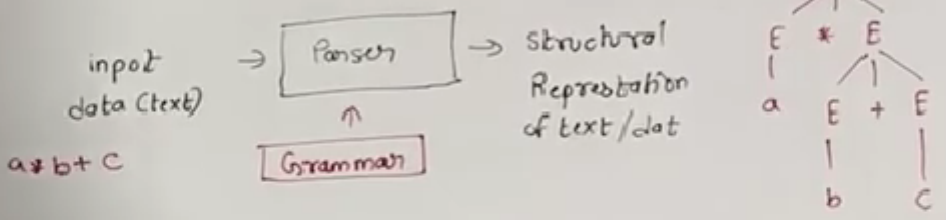
Gyan Pur





Natural language for communication 1. Text Embeddings 2. Dialogue and Conversation 3. Sentiment Analysis 4. Text Summarisation

Parser



- It is a software component design for taking input data (text) and give structural representation of the input after checking for correct syntax or grammar
- How it is used in NLP
  - Grammar checking
  - Intermediate stage of Smantic Analysis

Basic concept of Grammer and Parse tree (CFG)

- mathematically a grammer G can be written as four tuples

(N, T, S, P)

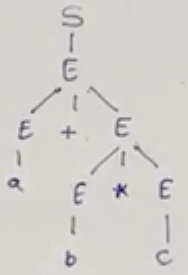
$V \quad \Sigma$

N → Non terminal
T → terminal
S → start Symbol
P → Production rules

Ex: $S \rightarrow E$

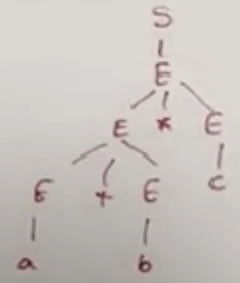
$E \rightarrow E+E \mid E*E \mid a \mid b \mid c$

input: a+b*c



Concept of Parse tree

- It is Graphical Representation of Derivation
- Start Symbol is root of Parse tree
- Leaf nodes are terminals
- Interior nodes are non-terminal
- If Parse Properly will create a input text



Parsing in NLP

I/p:- The little boy ran quickly

<Sentence> → <noun phrase> <verb Phrase>

<noun Phrase> → <adjective> <noun Phrase> | <adjective> <singular noun>

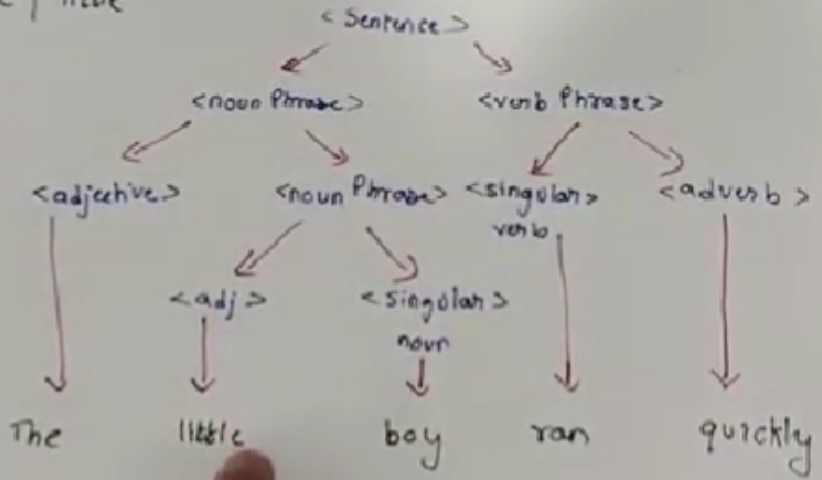
<verb Phrase> → <singular verb> <adverb>

<adjective> → a | the | little

<singular noun> → boy

<singular verb> → ran

<adverb> → quickly



Grammar

S → VP

VP → Verb NP

NP → Det Nom

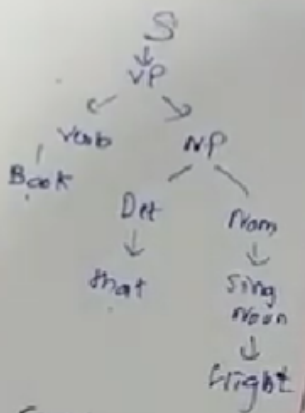
Det → that

Nom → singular noun

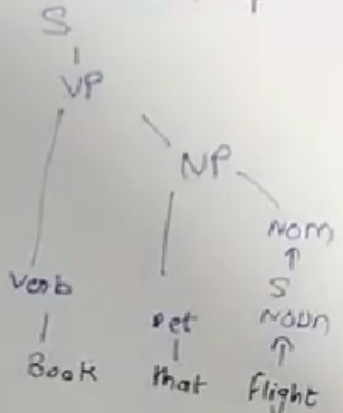
Verb → Book

S Noun → Flight

Top down



Bottom up



Input: Book that Flight

Augmented Grammar

If G is a grammar with start symbol S, then, the augmented grammar for G is G' with a new start symbol S' and production S' → S will be included.

The purpose of this new starting production is to indicate to the parser when it should stop parsing and announce acceptance of input. This would occure when the parser was about to reduce by S' → S.

Given grammar G:

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid id$$

So corresponding Augmented Grammar G':

$$E' \rightarrow E$$
$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid id$$

Semantic Analysis

- NLP is a field which aims to give machine the ability to understand natural language
- Semantic analysis is a sub topic of it
- Semantic analysis means giving exact meaning of text.
- Role of semantic analyser is to check text for meaningfulness

Semantic Analysis

Meaning of individual word
(lexical semantics)

Combination of words
(Sentences)

- Relationship exist between words
- The word order
- Context
- Semantic structure
- Real world knowledge

Composition Semantics

- It involves how word combines to form Larger meaning
- It says that meaning of each word matters but Syntax and way in which the Sentence are constructed Plays an important role as well

- Example: I like you and you like me are same words but meaning are different

I → subject	you → subject
you → object	me → object

- Lexical Semantic + Composition Semantic = Semantic Analysis
- This is called Principle of Compositionality

Basic Building blocks of Semantic System

Entities:- It represents the individual such as particular person, location, etc Example: Haryana, Rampyaari;

Concept:- It represent general category of individuals like person, vehicle, animals, etc.

Relations :- Relation between entities and concept
example:- Rampyaari is a vehicle

Predicates :- It represent verb structure, example: case grammer and semantic roles

. Case grammar: A form of Grammer in which Structure of Sentence is analyse.

Approaches

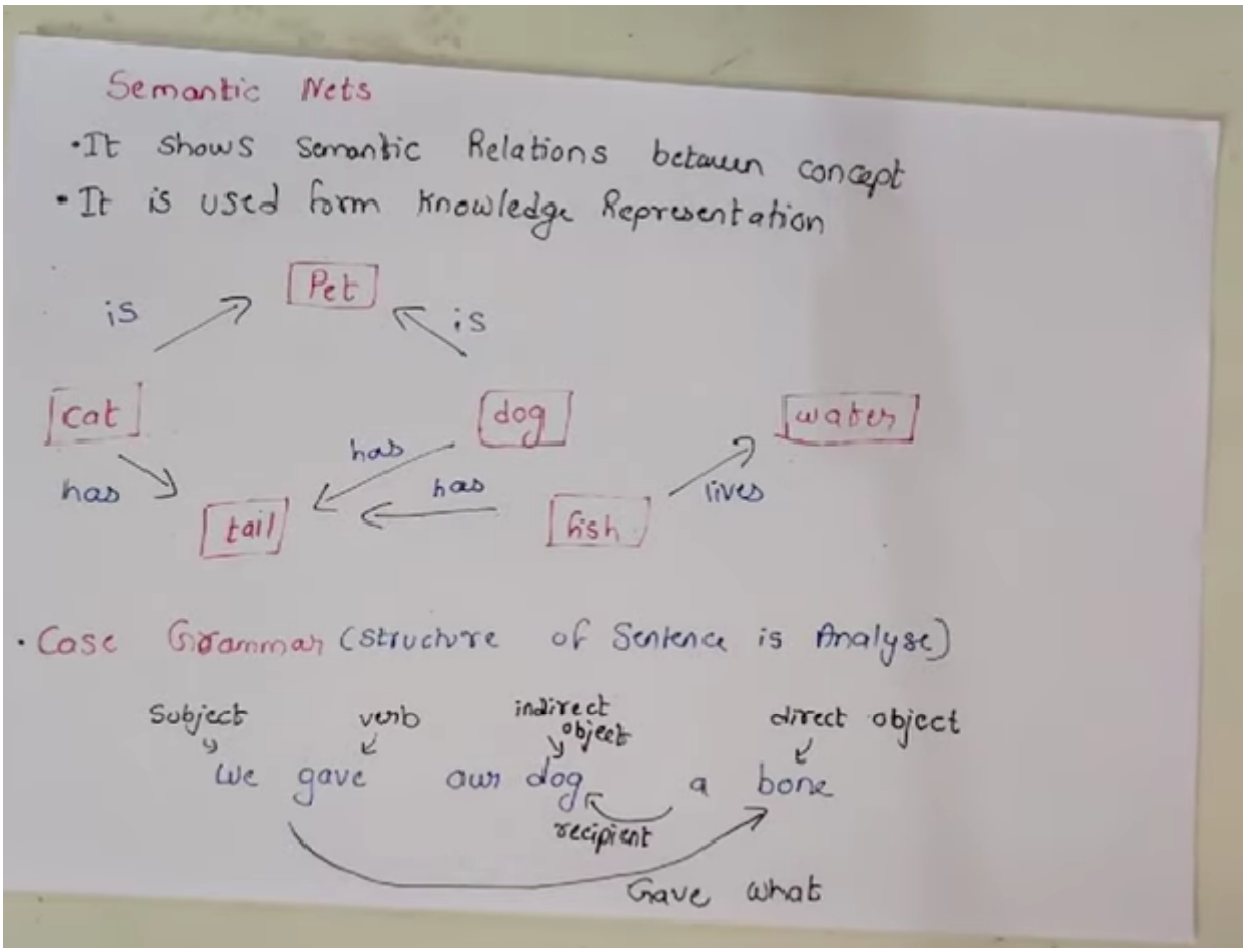
- First order Predicate logic
- Semantic Nets
- Case Grammer

1. FOPL → It convert sentence into logical form

Ex: Jack loves jill → loves (jack, jill)

Ex 2 : Sumer takes Engineering or Pharmacy

takes (Sumer, Engineering) V takes (Sumer, Pharmacy)


Semantic Nets
•It shows Semantic Relations between concept
•It is used form knowledge Representation
is
Pet
is
cat
dog
water
has
has
has
tail
fish
lives
•Case Grammar (structure of Sentence is Analyse)
Subject
verb
indirect object
direct object
We gave our dog a bone
recipient
Gave what

Expert Systems

The Expert Systems are the computer applications developed to solve complex problems in a particular domain, at the level of extra-ordinary human intelligence and expertise.

Characteristics of E.S:

- → High performance
- → Understandable
- → Reliable
- → Highly responsive

Capabilities of Expert System

Expert Systems are capable of -

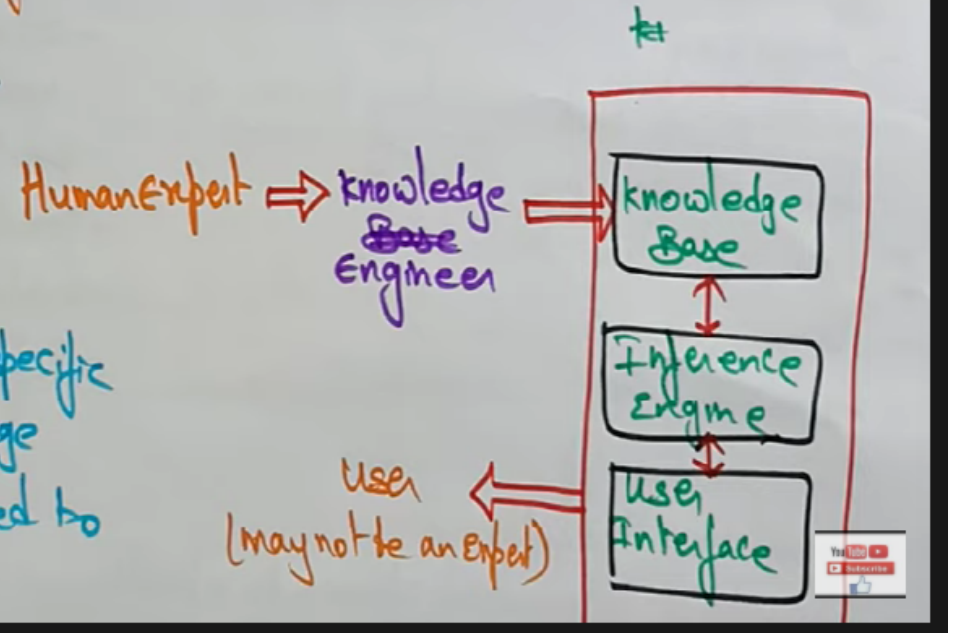
- Advising
- instructing and assisting human in decision making
- Demonstrating
- Deriving a solution
- Explaining
- Interpreting i/p
- Predicting result
- Suggesting alternative options to a problem

Incapable of

- Substituting human decision makers
- Possessing human capabilities
- Producing accurate o/p for inadequate knowledge base
- Refining their own knowledge

The components of ES include.

- Knowledge Base
- Inference Engine
- User Interface



Knowledge Base:

It contains domain-specific & high quality knowledge. Knowledge is required to exhibit intelligence.

The success of any ES majorly depends upon the collection of highly accurate and precise knowledge.

Inference Engine:

It is the brain of expert system. It contains rules to solve a specific problem. It selects facts and rules to apply when to answer the user's query. It provides reasoning about the information in the knowledge base. Inference Engine helps in deducing the problem to find the solution. This component is also helpful for formulating conclusions

User Interface:

The user interface is the most Coucial part of the exp system. This component takes the user's query in a readable form and passes it to the inference engine.

- After that, it displays the results to the user. In other wor its an interface that helps the user communicate with the expert system.

The Process of Building An Expert System

- Determining the characteristics of the problem
- knowledge engineer and domain expert work in coherence to define the problem.

Determining the characteristics of the

knowledge engineer and domain expert work in coherence to define the problem. human expert E.g: knowledge engineer mk knowledge Base computer understandable

the knowledge engineer translates the knowledge into a computer understandable lang. interpreting He designs an inference engine a reasoning structure, which can use knowledge when need knowledge expert also determines how to integrate the use of uncertain knowledge in the reasoning process and what type of explanation would be useful.

- Information management
- Hospitals & medical facilities
- Help desks management
- Employee performance evaluation.
- Loan analysis
- Virus detection
- Warehouse optimization
- planning & scheduling
- Stock market trading
- Airline scheduling & Cargo scheduling
- Process monitoring & control