UNIT 5: Knowledge & Reasoning

PART - B

Predicate Logic

Also known as

First-order logic, quantificational logic, first-order predicate calculus

- In this topic, we begin exploring one particular way of representing facts — the language of logic.
- The logical formalism is appealing because it immediately suggests a powerful way of deriving new knowledge from old — mathematical deduction.
- In this formalism, we can conclude that a new statement is true by proving that it follows from the statements that are already known.
- Thus the idea of a proof, as developed in mathematics as a rigorous way of demonstrating the truth of an already believed proposition, can be extended to include deduction as a way of deriving answers to questions and solutions to problems.

What is Predicate Logic?

- It is a **collection** of formal systems used in mathematics, philosophy, linguistics, and computer science.
- First-order logic uses **quantified** variables over non-logical objects, and allows the use of sentences that contain variables, so that rather than propositions such as "**Socrates is a man**", one can have expressions in the form "**there exists x such that x is Socrates and x is a man**", where "**there exists**" is a quantifier, while **x** is a variable.
- This **distinguishes** it from propositional logic, which does not use quantifiers or relations; in this sense, propositional logic is the foundation of first-order logic.

First-order logic

- First-order logic is another way of **knowledge representation** in artificial intelligence. It is an extension to propositional logic.
- FOL is sufficiently **expressive** to represent the natural language statements in a concise way.
- First-order logic is also known as **Predicate logic** or **First-order predicate logic**.
- First-order logic is a powerful language that develops information about the **objects** in a more easy way and can also express the **relationship** between those objects.
- First-order logic (like natural language) does not only assume that the world contains **facts** like propositional logic but also assumes the following things in the world:
  - **Objects**: A, B, people, numbers, colors, wars, theories, squares, pits, ...
  - **Relations**: It can be **unary** relation such as: red, round, is adjacent, or **n-ary** relation such as: the sister of, brother of, has color, comes between
  - **Function**: Father of, best friend, third inning of, end of, ...
- As a natural language, first-order logic has **two** main parts:
  - Syntax
  - Semantics

Proposition Vs. Predicate Logic

- The predicate refers to the property that the subject of the statement can take on.

$$\underbrace{i^2 + 3k}_{\textit{subject}} \underbrace{\geq 10 + j}_{\textit{predicate}}$$

- We can assign values to each variable — thus, creating a true or false proposition, as seen in the example below.

Question:

$P(x): x + y \geq 6$

Possible Solutions:

Let $P(7,1)$ $\quad P(7,1): \ (7)+(1) \geq 6$ $\quad$ True propositional statement

$8 \geq 6$

Let $P(3,2)$ $\quad P(3,2): \ (3)+(2) \geq 6$ $\quad$ False propositional statement

$5 \not\geq 6$

Proposition Vs. Predicate Logic – Cont'd

- But this isn't **always effective** or **helpful**, as we ultimately want the predicate to be factual **over a range** of elements, not just the ones that we've hand-selected.
- How do we do this?
- We use **quantifiers** to create propositional functions. This process is called **quantification**.
- Quantifiers express the extent to which a predicate is **true over a range** of elements.
- For example, imagine we have the statement: "Every person who is 18 years of age or older is able to vote. Sarah is 18 years old."
- While it would seem logical to conclude that Sarah would then be able to vote legally, propositional logic alone is ill-equipped with handling quantified variables, namely, what does "every person" really mean?
- As the range of possibilities is too broad, we need to apply one of two types of **quantifiers** to help us achieve our goal:
  - **Universal Quantifier:** ∀
  - **Existential Quantifier:** ∃

Basic Elements of First-order logic

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf, ...
Connectives	∧, ∨, ¬, ⇒, ⇔
Equality	==
Quantifier	∀, ∃

Points to remember:

The main connective for universal quantifier $\forall$ is implication $\rightarrow$.

The main connective for existential quantifier $\exists$ is and $\wedge$.

Universal Quantifier

- Universal Quantification is the proposition that a property is true for all the values of a variable in a particular domain, sometimes called the domain of discourse or the universe of discourse.
- Usually, universal quantification takes on any of the following forms:
  - P(x) is true for all values of x
  - For all x, P(x)
  - For each x, P(x)
  - For every x, P(x)
  - Given any x, P(x)
- And is symbolically denoted

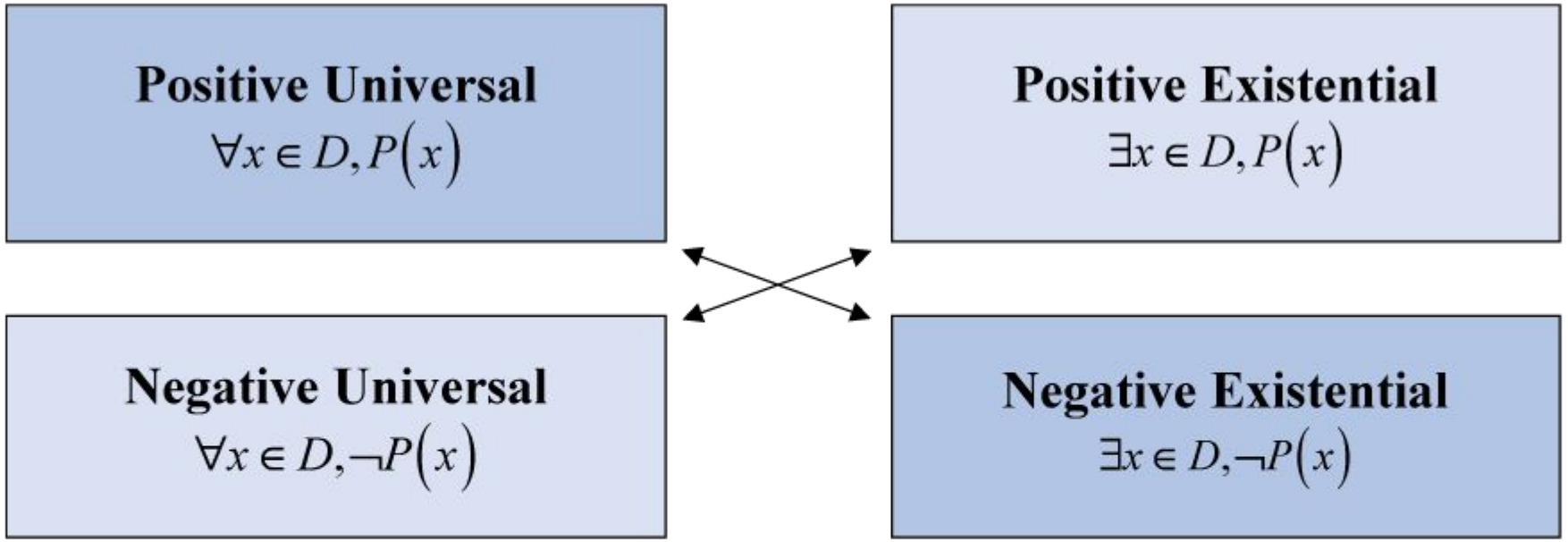
$\forall x \in D, P(x)$ " for all x, belonging to domain D, p(x) is true"

Existential Quantifier

- Existential quantification is the proposition that a property is true for some value in a particular domain.
- Customarily, existential quantification takes on one of the following forms:
  - There exists an x such that P(x)
  - There exists an element x in the domain such that P(x)
  - For some x, P(x)
  - There is some x such that P(x)
- And is symbolically denoted

$\exists x \in D, P(x)$ "there exists x, belonging to domain D, such that p(x) is true"

Universal vs. Existential Predicate



$$\neg\left(\forall x \in D, P(x)\right) \equiv \exists x \in D, \neg P(x)$$

$$\neg\left(\exists x \in D, P(x)\right) \equiv \forall x \in D, \neg P(x)$$

Some Examples of FOL using quantifier

- All birds fly.

  $\forall$ **x bird(x) $\rightarrow$fly(x)**.

- Every man respects his parent.

  $\forall$ **x man(x) $\rightarrow$ respects (x, parent)**.

- Some boys play cricket.

  $\exists$ **x boys(x) $\wedge$ play(x, cricket)**.

- Not all students like both Mathematics and Science.

  $\neg\forall$ **(x) [student(x) $\rightarrow$ like(x, Mathematics) $\wedge$ like(x, Science)].**

WFF - Well Formed Formula

- Every wizard who is not Voldemort is mortal

  $\forall x$ (**(WIZARD(x)** $\wedge$ $\neg$**x = v)** $\rightarrow$ **MORTAL(x)**)

  For every entity **x**, if **x** is a wizard and **x** is not equal to Voldemort, then **x** is mortal

CAT(x) The **1-place** predicate CAT and the variable **x** combine to form a wff.

x is a cat

B. LIKE(x,s) The **2-place** predicate LIKE combines with the variable **x** and individual **s** to form a wff.

x liked Sue

C. LIKE(x,y) The 2-place predicate LIKE combines with the variables **x** and **y** to form a wff.

x liked y

Examples: Predicate logic / First order logic

1. Some green dragon is sleeping
2. No green dragon is sleeping
3. Every green dragon is sleeping
4. Not every green dragon is sleeping

1. $\exists x ((G(x) \wedge D(x)) \wedge S(x))$
2. $\neg \exists x ((G(x) \wedge D(x)) \wedge S(x))$
3. $\forall x ((G(x) \wedge D(x)) \rightarrow S(x))$
4. $\neg \forall x ((G(x) \wedge D(x)) \rightarrow S(x))$

1. Some dragon is sleeping or twitching
2. No dragon is sleeping or twitching
3. Every dragon is sleeping or twitching
4. Not every dragon is sleeping or twitching

1. $\exists x [D(x) \wedge (S(x) \vee T(x))]$
2. $\neg \exists x [D(x) \wedge (S(x) \vee T(x))]$
3. $\forall x [D(x) \rightarrow (S(x) \vee T(x))]$
4. $\neg \forall x [D(x) \rightarrow (S(x) \vee T(x))]$

Using Predicate Logic

- Marcus was a man.

  man(Marcus)

- Marcus was a Pompeian.

  Pompeian(Marcus)

- All Pompeians were Romans.

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

- Caesar was a ruler.

  ruler(Caesar)

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

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

Representing INSTANCE and ISA Relationships

- The **first part** of the figure contains the representations we have already discussed. In these representations, class membership is represented with **unary predicates** (such as Roman), each of which corresponds to a class.
- The **second part** of the figure contains representations that use the **instance** predicate explicitly.
- The **third part** contains representations that use both the instance and isa predicates explicitly. The use of the **isa** predicate simplifies the representation of sentence 3, but it requires that one additional axiom (shown here as number 6) be provided.

1. Man(Marcus).
2. Pompeian(Marcus).
3. $\forall x$: Pompeian(x) $\rightarrow$ Roman(x).
4. ruler(Caesar).
5. $\forall x$: Roman(x) $\rightarrow$ loyalto(x, Caesar) $\vee$ hate(x, Caesar).

1. instance(Marcus, man).
2. instance(Marcus, Pompeian).
3. $\forall x$: instance(x, Pompeian) $\rightarrow$ instance(x, Roman).
4. instance(Caesar, ruler).
5. $\forall x$: instance(x, Roman). $\rightarrow$ loyalto(x, Caesar) $\vee$ hate(x, Caesar).

1. instance(Marcus, man).
2. instance(Marcus, Pompeian).
3. isa(Pompeian, Roman)
4. instance(Caesar, ruler).
5. $\forall x$: instance(x, Roman). $\rightarrow$ loyalto(x, Caesar) $\vee$ hate(x, Caesar).
6. $\forall x$: $\forall y$: $\forall z$: instance(x, y) $\wedge$ isa(y, z)$\rightarrow$ instance(x, z).

Computable Functions and Predicates

- To express simple facts, such as the following greater-than and less-than relationships:

  gt(1,0) It(0,1) gt(2,1) It(1,2) gt(3,2) It(2,3)
- It is often also useful to have computable functions as well as computable predicates.
- Thus we might want to be able to evaluate the truth of gt(2 + 3,1)
- To do so requires that we first compute the value of the plus function given the arguments 2 and 3, and then send the arguments 5 and 1 to gt.

Consider the following set of facts, again involving Marcus:

1) Marcus was a man.

man(Marcus)

2) Marcus was a Pompeian.

Pompeian(Marcus)

3) Marcus was born in 40 A.D.

born(Marcus, 40)

4) All men are mortal.

x: man(x) → mortal(x)

5) All Pompeians died when the volcano erupted in 79 A.D.

erupted(volcano, 79) ∧ ∀ x : [Pompeian(x) → died(x, 79)]

6) No mortal lives longer than 150 years.

∀x: ∀t1: ∀t2: mortal(x) ∧ born(x, t1) ∧ gt(t2 – t1,150) → died(x, t2)

7) It is now 2022.

now = 1991

So, Now suppose we want to answer the question **"Is Marcus alive?"**

• The statements suggested here, there may be two ways of deducing an answer.

• Either we can show that Marcus is dead because he was killed by the volcano or we can show that he must be dead because he would otherwise be more than 150 years old, which we know is not possible.

• Also, As soon as we attempt to follow either of those paths rigorously, however, we discover, just as we did in the last example, that we need some additional knowledge. For example, our statements talk about dying, but they say nothing that relates to being alive, which is what the question is asking.

So we add the following facts:

8) Alive means not dead.

∀ x: ∀ t: [alive(x, t) → ¬ dead(x, t)] ∧ [¬ dead(x, t) → alive(x, t)]

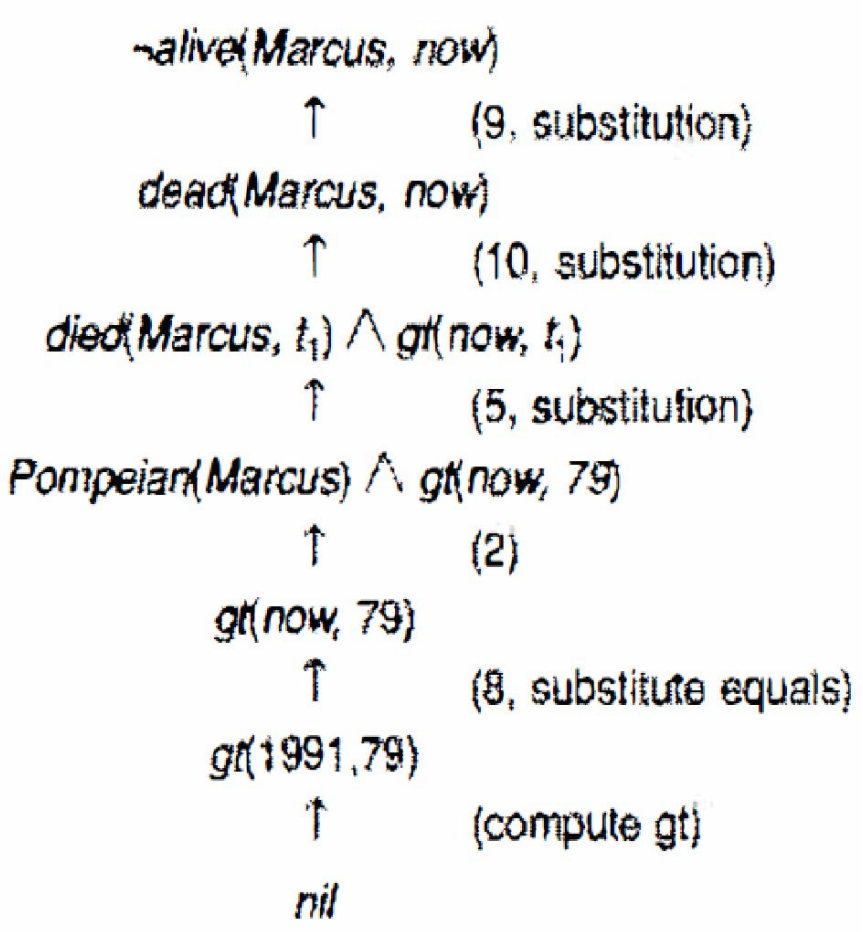
9) If someone dies, then he is dead at all later times.

∀ x: ∀ t1: ∀ t2: died(x, t1) ∧ gt(t2, t1) → dead(x, t2)

So, based on these facts, we can answer the question "Is Marcus alive?" by proving: **¬ alive(Marcus, now)**

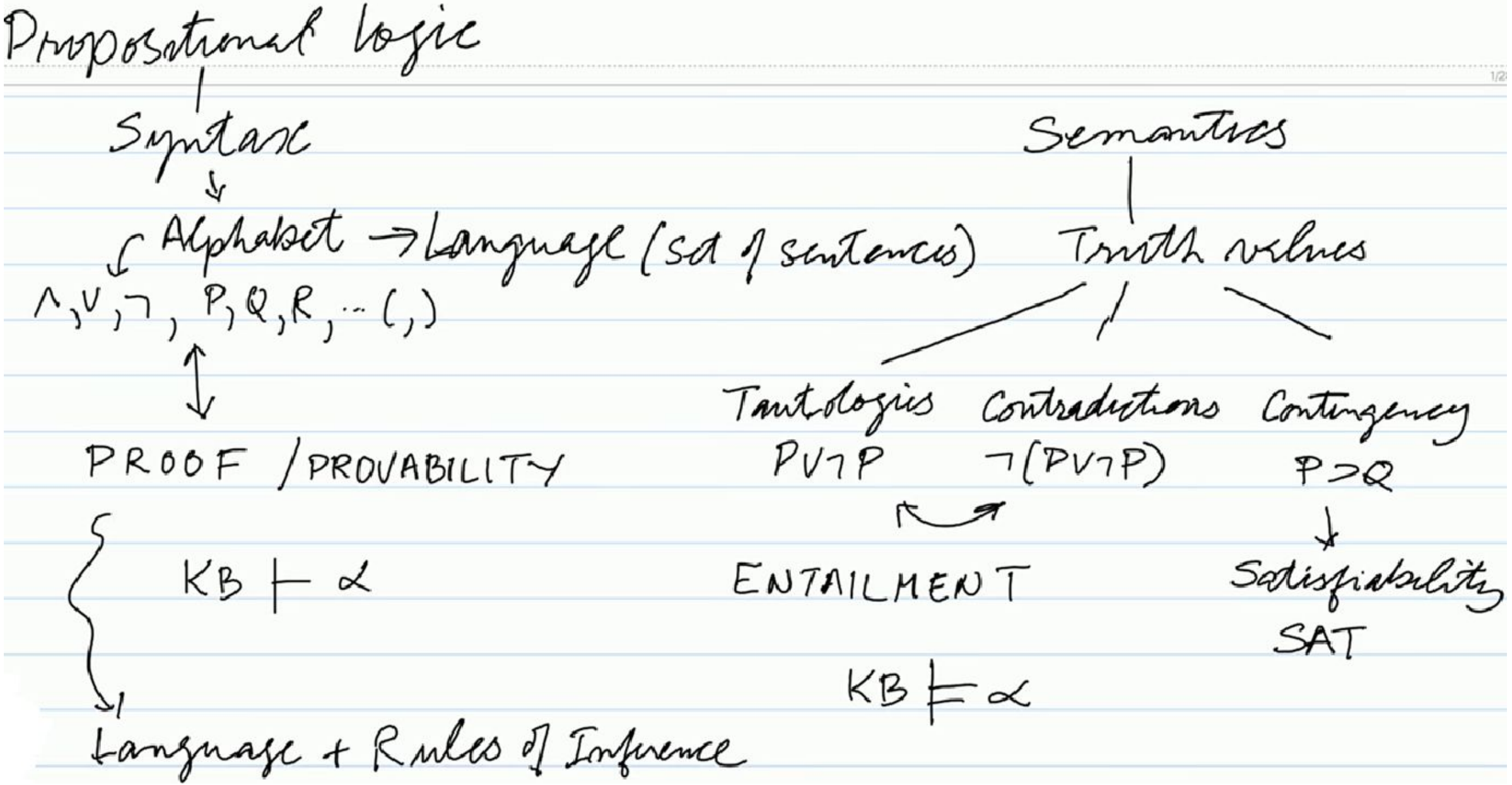
One way of proving that Marcus is not alive…

1. **man(Marcus)**
2. **Pompeian(Marcus)**
3. **born(Marcus, 40)**
4. **x: man(x) → mortal(x)**
5. **∀ x : Pompeian(x) → died(x, 79)**
6. **erupted(volcano, 79)**
7. **∀x: ∀t1: ∀t2: mortal(x) ∧ born(x, t1) ∧ gt(t2 – t1,150) → died(x, t2)**
8. **now = 1991**
9. **∀ x: ∀ t: [alive(x, t) → ¬ dead(x, t)] ∧ [¬ dead(x, t) → alive(x, t)]**
10. **∀ x: ∀ t1: ∀ t2: died(x, t1) ∧ gt(t2, t1) → dead(x, t2)**



Natural Deduction

- Testing whether a proposition is a tautology by testing every possible truth assignment is expensive—there are exponentially many. We need a deductive system, which will allow us to construct proofs of tautologies in a step-by-step fashion.
- The system we will use is known as natural deduction. The system consists of a set of rules of inference for deriving consequences from premises. One builds a proof tree whose root is the proposition to be proved and whose leaves are the initial assumptions or axioms (for proof trees, we usually draw the root at the bottom and the leaves at the top).
- For example, one rule of our system is known as modus ponens. Intuitively, this says that if we know P is true, and we know that P implies Q, then we can conclude Q.


Propositional logic
Syntax
Alphabet → Language (set of sentences)
∧, ∨, ¬, P, Q, R, ... (,)
PROOF / PROVABILITY
KB ⊢ α
Language + Rules of Inference
Semantics
Truth values
Tautologies
PV¬P
Contradictions
¬(PV¬P)
Contingency
P⇒Q
ENTAILMENT
KB ⊨ α
Satisfiability
SAT

Proof systems

MODUS PONENS

MP: $P, P \supset Q \vdash Q$

MP is valid because

$((P \wedge (P \supset Q)) \supset Q)$

$$\frac{P \qquad P \supset Q}{Q}$$

PROPOSITIONAL VARIABLES

$$\frac{(A \wedge B) \vee C \qquad ((A \wedge B) \vee C) \supset RVS}{RVS}$$

$$\frac{Sentence1 \qquad Sentence1 \supset Sentence2}{Sentence2}$$

Pick some rule
with matching antecedents
Add the consequent to KB

until some termination criteria

Termination:

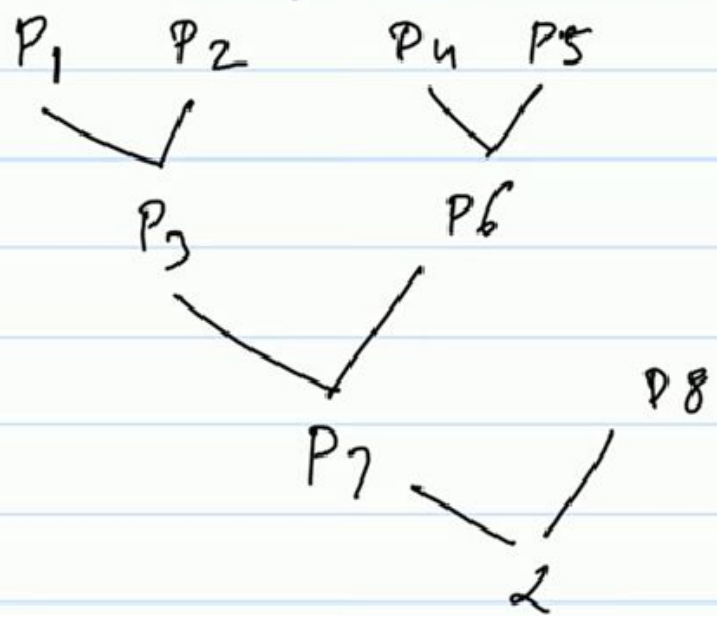
Direct Proof / Natural Deduction

↓

"until α is added to KB"

GENTZEN

Proof Tree



Indirect Proof

↓

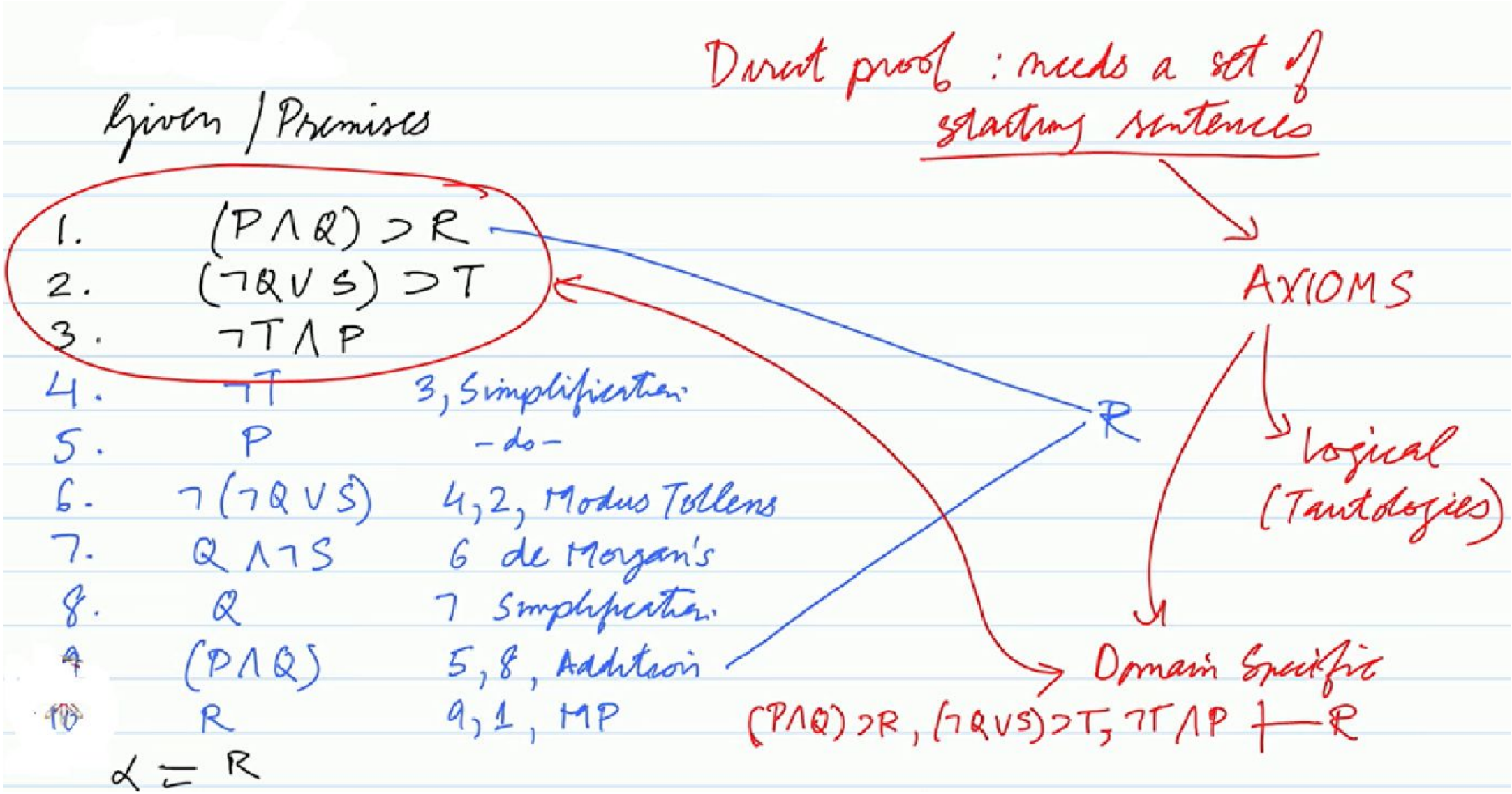
"proof by contradiction."

↓

If you add negation of goal

$\{\neg\alpha\} \cup KB \vdash \perp$

Natural Deduction: Example



Inference in Predicate Logic

Also known as

First-order logic, quantificational logic, first-order predicate calculus

- Inference in First-Order Logic is used to deduce new facts or sentences from existing sentences.
- Before understanding the FOL inference rule, let's understand some basic terminologies used in FOL.

Substitution

- Substitution is a fundamental operation performed on terms and formulas.
- It occurs in all inference systems in first-order logic.
- The substitution is complex in the presence of quantifiers in FOL.
- If we write **F[a/x]**, so it refers to substitute a constant "**a**" in place of variable "**x**".

Equality

- First-Order logic does not only use predicate and terms for making atomic sentences but also uses another way, which is equality in FOL.
- For this, we can use **equality symbols** which specify that the two terms refer to the same object.
- **Example: Brother (John) = Smith.**
- As in the above example, the object referred by the **Brother (John)** is similar to the object referred by **Smith**.
- The equality symbol can also be used with negation to represent that two terms are not the same objects.
- **Example: ¬ (x = y) which is equivalent to x ≠ y.**

FOL inference rules for quantifier

There are four quantifier rules of inference that allow you to remove or introduce a quantifier

Rule of Inference	Name
$\therefore \dfrac{\forall x P(x)}{P(c)}$	Universal instantiation
$\therefore \dfrac{P(c) \text{ for an arbitrary } c}{\forall x P(x)}$	Universal generalization
$\therefore \dfrac{\exists x P(x)}{P(c) \text{ for some element } c}$	Existential instantiation
$\therefore \dfrac{P(c) \text{ for some element } c}{\exists x P(x)}$	Existencial generalization

1. Universal Generalization

- Universal generalization is a valid inference rule which states that if premise P(c) is true for any arbitrary element c in the universe of discourse, then we can have a conclusion as $\forall$ x P(x).
- It can be represented as: . $\frac{P(c)}{\forall x\, P(x)}$
- This rule can be used if we want to show that every element has a similar property.
- In this rule, x must not appear as a free variable.
- **Example:** 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.

2. Universal Instantiation

- Universal instantiation is also called as universal elimination or UI is a valid inference rule. It can be applied multiple times to add new sentences.
- The new KB is logically equivalent to the previous KB.
- As per UI, **we can infer any sentence obtained by substituting a ground term for the variable**.
- The UI rule state that we can infer any sentence P(c) by substituting a ground term c (a constant within domain x) from $\forall$ **x P(x) for any object in the universe of discourse**.

It can be represented as:.

- **Example:1.**

  IF "Every person like ice-cream"=> $\forall$ x P(x) so we can infer that
  "John likes ice-cream" => P(c)

- **Example: 2.**

  Let's take a famous example,

  "All kings who are greedy are Evil." So let our knowledge base contains this detail as in the form of FOL:

- **$\forall$ x king(x) $\wedge$ greedy (x) $\rightarrow$ Evil (x),**

  So from this information, we can infer any of the following statements using Universal Instantiation:

- **King(John) $\wedge$ Greedy (John) $\rightarrow$ Evil (John),**
- **King(Richard) $\wedge$ Greedy (Richard) $\rightarrow$ Evil (Richard),**
- **King(Father(John)) $\wedge$ Greedy (Father(John)) $\rightarrow$ Evil (Father(John)),**

3. Existential Instantiation

- Existential instantiation is also called as Existential Elimination, which is a valid inference rule in first-order logic.
- It can be applied only once to replace the existential sentence.
- The new KB is not logically equivalent to old KB, but it will be satisfiable if old KB was satisfiable.
- This rule states that one can infer P(c) from the formula given in the form of $\exists x\ P(x)$ for a new constant symbol c.
- The restriction with this rule is that c used in the rule must be a new term for which P(c) is true.
- It can be represented as: $\frac{\exists x\ P(x)}{P(c)}$

- **Example:**
- From the given sentence: **$\exists$x Crown(x) $\wedge$ OnHead(x, John),**
- So we can infer: **Crown(K) $\wedge$ OnHead(K, John),** as long as K does not appear in the knowledge base.
- The above used K is a constant symbol, which is called **Skolem constant**.
- The Existential instantiation is a special case of **Skolemization process**.

4. Existential generalization/introduction

- An existential introduction is also known as an existential generalization, which is a valid inference rule in first-order logic.
- 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 the property P.
- It can be represented as: $\frac{P(c)}{\exists x P(x)}$
- **Example: Let's say that,**
  "Priyanka got good marks in English."
  "Therefore, someone got good marks in English."

Generalized Modus Ponens Rule

- For the inference process in FOL, we have a single inference rule which is called Generalized Modus Ponens. It is lifted version of Modus ponens.
- Generalized Modus Ponens can be summarized as, " P implies Q and P is asserted to be true, therefore Q must be True."
- According to Modus Ponens, for atomic sentences **pi, pi', q**. Where there is a substitution θ such that SUBST **(θ, pi',) = SUBST(θ, pi)**, it can be represented as:

$$\frac{p1', p2', \ldots, pn', (p1 \wedge p2 \wedge \ldots \wedge pn \Rightarrow q)}{SUBST(\theta, q)}$$

Example

- **We will use this rule for Kings are evil, so we will find some x such that x is king, and x is greedy so we can infer that x is evil.**

- Here let say, p1' is king(John) p1 is king(x)
- p2' is Greedy(y) p2 is Greedy(x)
- θ is {x/John, y/John} q is evil(x)
- SUBST(θ,q).

- This is known as UNIFICATION.

What is Unification?

- Unification is a process of making two different logical atomic expressions identical by finding a substitution. Unification depends on the substitution process.
- It takes two literals as input and makes them identical using substitution.
- Let Ψ_1 and Ψ_2 be two atomic sentences and σ be a unifier such that, $\boldsymbol{\Psi_1\sigma = \Psi_2\sigma}$, then it can be expressed as $\mathbf{UNIFY(\Psi_1, \Psi_2)}$.

- **Example: Find the MGU for Unify{King(x), King(John)}**
- Let Ψ_1 = King(x), Ψ_2 = King(John),

Substitution θ = {John/x} is a unifier for these atoms and applying this substitution, and both expressions will be identical.

- The UNIFY algorithm is used for unification, which takes two atomic sentences and returns a unifier for those sentences (If any exist).
- Unification is a key component of all first-order inference algorithms.
- It returns fail if the expressions do not match with each other.
- The substitution variables are called **Most General Unifier** or **MGU**.

Unification Example

Ψ_1	Ψ_2	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y,OJ)	{x/OJ,y/John}
Knows(John,x)	Knows(y,Mother(y))	{y/John,x/Mother(John)}
Knows(John,x)	Knows(x,OJ)	{fail}

- **E.g.** Let's say there are two different expressions, **P(x, y), and P(a, f(z))**.
- In this example, we need to make both above statements identical to each other. For this, we will perform the substitution.
- P(x, y)..... (i)
  P(a, f(z))..... (ii)
- Substitute x with a, and y with f(z) in the first expression, and it will be represented as **a/x** and f(z)/y.
- With both the substitutions, the first expression will be identical to the second expression and the substitution set will be: **[a/x, f(z)/y]**.

Conditions for Unification:

- **Following are some basic conditions for unification:**
- Predicate symbol must be same, atoms or expression with different predicate symbol can never be unified.
- Number of Arguments in both expressions must be identical.
- Unification will fail if there are two similar variables present in the same expression.

Unification Algorithm:

Step. 1: If Ψ_1 or Ψ_2 is a variable or constant, then:

a) If Ψ_1 or Ψ_2 are identical, then return NIL.

b) Else if $\Psi_1$is a variable,

a. then if Ψ_1 occurs in Ψ_2, then return FAILURE

b. Else return { (Ψ_2/ Ψ_1)}.

c) Else if Ψ_2 is a variable,

a. If Ψ_2 occurs in Ψ_1 then return FAILURE,

b. Else return {(Ψ_1/ Ψ_2)}.

d) Else return FAILURE.

Step.2: If the initial Predicate symbol in Ψ_1 and Ψ_2 are not same, then return FAILURE.

Step. 3: IF Ψ_1 and Ψ_2 have a different number of arguments, then return FAILURE.

Step. 4: Set Substitution set(SUBST) to NIL.

Step. 5: For i=1 to the number of elements in Ψ_1.

a) Call Unify function with the ith element of Ψ_1 and ith element of Ψ_2, and put the result into S.

b) If S = failure then returns Failure

c) If S ≠ NIL then do,

a. Apply S to the remainder of both L1 and L2.

b. SUBST= APPEND(S, SUBST).

Step.6: Return SUBST.

Most General Unifier

- **1. Find the MGU of {p(f(a), g(Y)) and p(X, X)}**
- Sol: S_0 => Here, Ψ_1 = p(f(a), g(Y)), and Ψ_2 = p(X, X)
  SUBST θ = {f(a) / X}
  S1 => Ψ_1 = p(f(a), g(Y)), and Ψ_2 = p(f(a), f(a))
  SUBST θ = {f(a) / g(y)},

  Unification failed.
- Unification is not possible for these expressions.

Most General Unifier

- **Find the MGU of {p(b, X, f(g(Z))) and p(Z, f(Y), f(Y))}**
- Here, Ψ_1 = p(b, X, f(g(Z))) , and Ψ_2 = p(Z, f(Y), f(Y))
  S_0 => { p(b, X, f(g(Z))); p(Z, f(Y), f(Y))}
  SUBST θ={b/Z}
- S_1 => { p(b, X, f(g(b))); p(b, f(Y), f(Y))}
  SUBST θ={f(Y) /X}
- S_2 => { p(b, f(Y), f(g(b))); p(b, f(Y), f(Y))}
  SUBST θ= {g(b) /Y}
- S_2 => { p(b, f(g(b)), f(g(b)); p(b, f(g(b)), f(g(b))} **Unified Successfully.**
  And Unifier = { b/Z, f(Y) /X , g(b) /Y}.

Most General Unifier

- **3. Find the MGU of {p (X, X), and p (Z, f(Z))}**
- Here, Ψ_1 = {p (X, X), and Ψ_2 = p (Z, f(Z))
  S_0 => {p (X, X), p (Z, f(Z))}
  SUBST θ= {X/Z}
  S1 => {p (Z, Z), p (Z, f(Z))}
  SUBST θ= {f(Z) / Z}, **Unification Failed**.

Most General Unifier

- **4. Find the MGU of UNIFY(prime (11), prime(y))**
- Here, Ψ_1 = {prime(11) , and Ψ_2 = prime(y)}
  S_0 => {prime(11) , prime(y)}
  SUBST θ= {11/y}
- S_1 => {prime(11) , prime(11)} , **Successfully unified.**
  Unifier: {11/y}.

Most General Unifier

- **5. Find the MGU of Q(a, g(x, a), f(y)), Q(a, g(f(b), a), x)}**
- Here, Ψ_1 = Q(a, g(x, a), f(y)), and Ψ_2 = Q(a, g(f(b), a), x)
  S_0 => {Q(a, g(x, a), f(y)); Q(a, g(f(b), a), x)}
  SUBST θ= {f(b)/x}
  S_1 => {Q(a, g(f(b), a), f(y)); Q(a, g(f(b), a), f(b))}
- SUBST θ= {b/y}
  S_1 => {Q(a, g(f(b), a), f(b)); Q(a, g(f(b), a), f(b))}, **Successfully Unified.**
- **Unifier: [a/a, f(b)/x, b/y].**

Most General Unifier

- **6. UNIFY(knows(Richard, x), knows(Richard, John))**
- Here, Ψ_1 = knows(Richard, x), and Ψ_2 = knows(Richard, John)
  S_0 => { knows(Richard, x); knows(Richard, John)}
  SUBST θ= {John/x}
  S_1 => { knows(Richard, John); knows(Richard, John)},
- **Successfully Unified.**
  Unifier: {John/x}.

Skolemization

- Conversion of sentences **FOL to CNF** requires skolemization.
- **Skolemization:** remove existential quantifiers by introducing new function symbols.
- **Special case:** introducing constants – **Skolem Contant**

Skolemization: Example

- Every philosopher writes at least one book.

  $\forall x[Philo(x) \rightarrow \exists y[Book(y) \wedge Write(x, y)]]$

- Eliminate Implication:

  $\forall x[\neg Philo(x) \vee \exists y[Book(y) \wedge Write(x, y)]]$

- Skolemize: substitute y by g(x) in order to remove $\exists$

  $\forall x[\neg Philo(x) \vee [Book(g(x)) \wedge Write(x, g(x))]]$

  here **g(x)** is **skolem** constant.

Prenex Normal Form

- It is often more convenient to deal with formulas in which all quantifiers have been moved to the front of the expression.
- These types of formulas are said to be in prenex normal form.

- **Definition:** A formula is in prenex normal form if it is of the form

$$Q_1x_1\ Q_2x_2\ \ldots\ Q_nx_nB$$

where $Q_i(i = 1, \ldots n)$ is $\forall$ or $\exists$ and the formula B is quantifier free.

Connection between $\forall$ and $\exists$

- $\forall x \neg \text{Likes}(x, \text{Chips}) \equiv \neg \exists x \text{Likes}(x, \text{Chips})$
- $\forall x \text{Likes}(x, \text{IceCream}) \equiv \neg \exists x \neg \text{Likes}(x, \text{IceCream})$

1. $\forall x \neg P \equiv \neg \exists x P$
2. $\neg \forall x P \equiv \exists x \neg P$
3. $\forall x P \equiv \neg \exists x \neg P$
4. $\exists x P \equiv \neg \forall x \neg P$

1. $\neg (P \vee Q) \equiv \neg P \wedge \neg Q$
2. $\neg (P \wedge Q) \equiv \neg P \vee \neg Q$
3. $P \wedge Q \equiv \neg (\neg P \vee \neg Q)$
4. $P \vee Q \equiv \neg (\neg P \wedge \neg Q)$

Resolution in Predicate Logic: Example

- All hounds howl at night.

  $\forall x\ (HOUND(x) \rightarrow HOWL(x))$

- Anyone who has any cats will not have any mice.

  $\forall x\ \forall y\ (HAVE\ (x,y) \wedge CAT\ (y) \rightarrow \neg\ \exists z\ (HAVE(x,z) \wedge MOUSE\ (z)))$

- Light sleepers do not have anything which howls at night.

  $\forall x\ (LS(x) \rightarrow \neg\ \exists y\ (HAVE\ (x,y) \wedge HOWL(y)))$

- John has either a cat or a hound.

  $\exists x\ (HAVE\ (John,x) \wedge (CAT(x) \vee HOUND(x)))$

- (Conclusion) If John is a light sleeper, then John does not have any mice.

  $LS(John) \rightarrow \neg\ \exists z\ (HAVE(John,z) \wedge MOUSE(z))$

The next step is to transform each wff into Prenex Normal Form, skolemize, and rewrite as clauses in conjunctive normal form; these transformations are shown below.

1. $\forall$ x (HOUND(x) $\rightarrow$ HOWL(x))

   $\neg$ ***HOUND(x)*** $\vee$ ***HOWL(x)***

2. $\forall$ x $\forall$ y (HAVE (x,y) $\wedge$ CAT (y) $\rightarrow \neg \exists$ z (HAVE(x,z) $\wedge$ MOUSE (z)))

   $\forall$ x $\forall$ y (HAVE (x,y) $\wedge$ CAT (y) $\rightarrow$ $\forall$ z $\neg$ (HAVE(x,z) $\wedge$ MOUSE (z)))

   $\forall$ x $\forall$ y $\forall$ z ($\neg$ (HAVE (x,y) $\wedge$ CAT (y)) $\vee \neg$ (HAVE(x,z) $\wedge$ MOUSE (z)))

   $\neg$ HAVE(x,y) $\vee \neg$ CAT(y) $\vee \neg$ HAVE(x,z) $\vee \neg$ MOUSE(z)

3. $\forall$ x (LS(x) $\rightarrow \neg \exists$ y (HAVE (x,y) $\wedge$ HOWL(y)))

   $\forall$ x (LS(x) $\rightarrow$ $\forall$ y $\neg$ (HAVE (x,y) $\wedge$ HOWL(y)))

   $\forall$ x $\forall$ y (LS(x) $\rightarrow \neg$ HAVE(x,y) $\vee \neg$ HOWL(y))

   $\forall$ x $\forall$ y ($\neg$ LS(x) $\vee \neg$ HAVE(x,y) $\vee \neg$ HOWL(y))

   $\neg$ LS(x) $\vee \neg$ HAVE(x,y) $\vee \neg$ HOWL(y)

4. $\exists$ x (HAVE (John,x) $\wedge$ (CAT(x) $\vee$ HOUND(x)))

HAVE(John,a) $\wedge$ (CAT(a) $\vee$ HOUND(a))

5. $\neg$ [LS(John) $\rightarrow \neg \exists$ z (HAVE(John,z) $\wedge$ MOUSE(z))] (negated conclusion)

$\neg$ [$\neg$ LS (John) $\vee \neg \exists$ z (HAVE (John, z) $\wedge$ MOUSE(z))]

LS(John) $\wedge \exists$ z (HAVE(John, z) $\wedge$ MOUSE(z)))

LS(John) $\wedge$ HAVE(John,b) $\wedge$ MOUSE(b)

1. $\neg$ HOUND(x) $\vee$ HOWL(x)
2. $\neg$ HAVE(x,y) $\vee \neg$ CAT(y) $\vee \neg$ HAVE(x,z) $\vee \neg$ MOUSE(z)
3. $\neg$ LS(x) $\vee \neg$ HAVE(x,y) $\vee \neg$ HOWL(y)
4.
   a) HAVE(John,a)
   b) CAT(a) $\vee$ HOUND(a)
5.
   a) LS(John)
   b) HAVE(John,b)
   c) MOUSE(b)

Now we proceed to prove the conclusion by resolution using the above clauses. Each result clause is numbered; the numbers of its parent clauses are shown to its left.

[1.,4.(b):]	6.	CAT(a) $\vee$ HOWL(a)
[2,5.(c):]	7.	$\neg$ HAVE(x,y) $\vee \neg$ CAT(y) $\vee \neg$ HAVE(x,b)
[7,5.(b):]	8.	$\neg$ HAVE(John,y) $\vee \neg$ CAT(y)
[6,8:]	9.	$\neg$ HAVE(John,a) $\vee$ HOWL(a)
[4.(a),9:]	10.	HOWL(a)
[3,10:]	11.	$\neg$ LS(x) $\vee \neg$ HAVE(x,a)
[4.(a),11:]	12.	$\neg$ LS(John)
[5.(a),12:]	13.	( )

Logic Programming

- Logic programming is a programming paradigm in which logical assertions are viewed as programs.
- These are several logic programming systems, PROLOG is one of them.
- ***A PROLOG program consists of several logical assertions where each is a horn clause i.e. a clause with at most one positive literal.***

- Ex : P, P V Q, P → Q
  - The facts are represented on Horn Clause for two reasons.
    - Because of a uniform representation, a simple and efficient interpreter can be written.
    - The logic of Horn Clause is decidable.

Consider the following example

Logical representation

- $\forall$ x : pet(x) ^ small (x) → apartmentpet(x)
- $\forall$ x : cat(x) $\vee$ dog(x) → pet(x)
- $\forall$ x : poodle (x) → dog (x) ^ small (x)
- poodle (fluffy)

Prolog representation

- apartmentpet (x) :- pet(x), small (x).
- pet (x) :- cat (x).
- pet (x) :- dog(x).
- dog(x) :- poodle (x).
- small (x) :- poodle(x).
- poodle (fluffy).

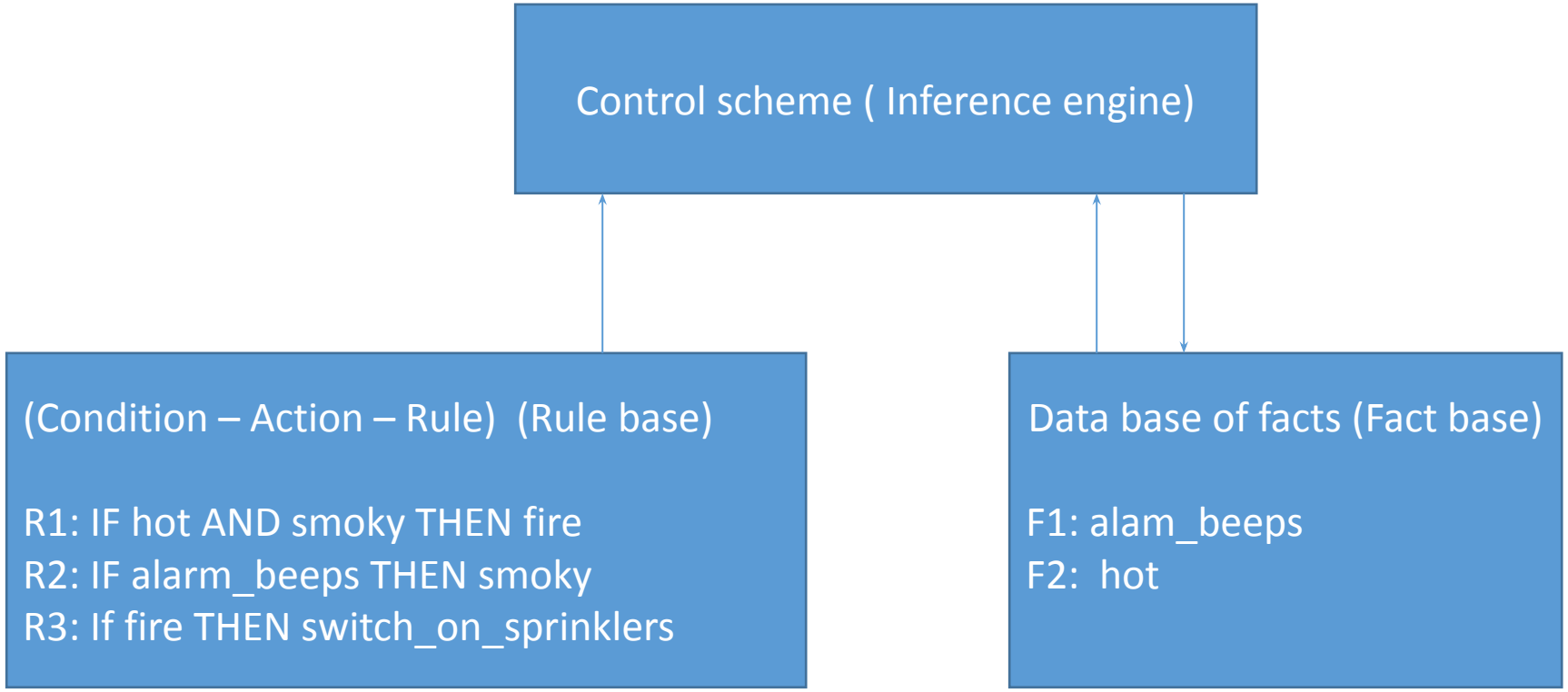
Rule-Based System Architecture

- **A collection of facts**
  - Assertions that represent domain specific knowledge.
    - E.g. Anil is hardworking.
- **A collection of rules**
  - assertions given in implicational form
    1. **Implication**: the '→' in the rule
    2. **Antecedent:** the part of the rule before '→'
    3. **Consequent:** the part of the rule after '→'
  - Read a rule as "antecedent implies consequent" or

    IF antecedent THEN consequent
- **An inference engine**

  "A generic control mechanism that applies the axiomatic knowledge present in the knowledge base to the task-specific data to arrive at some conclusion".

Rule-Based System Architecture



Two control strategies:

1. forward chaining
   - working from the facts to a conclusion.

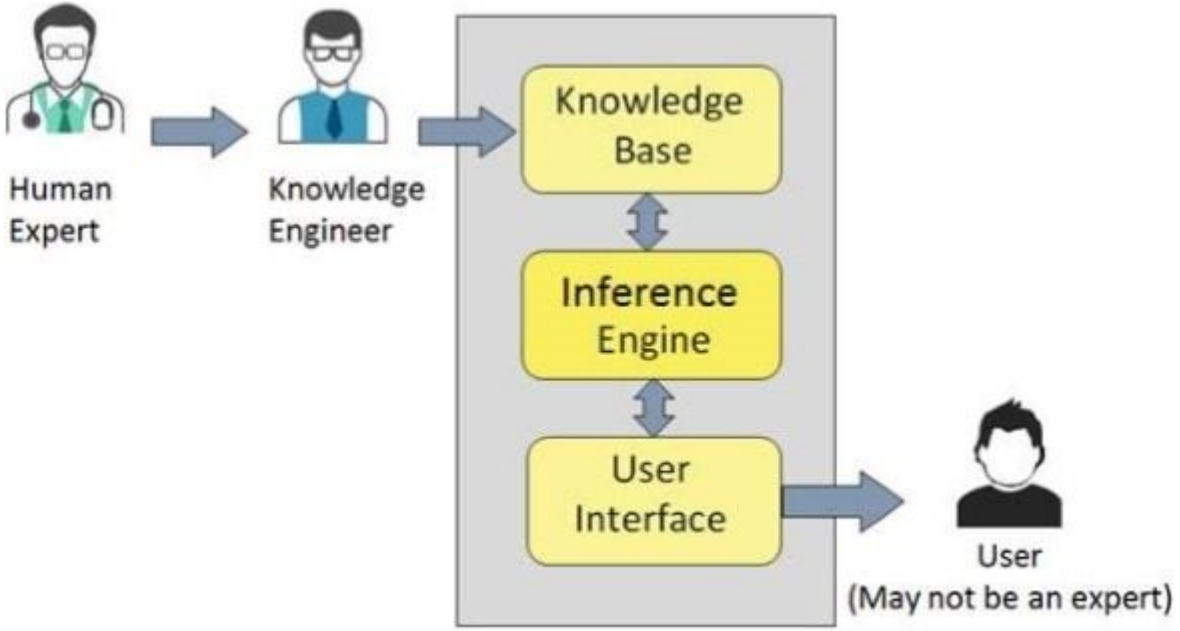
2. backward chaining
   - reasoning from goals back to facts

Introduction to the Expert System

- A brief overview of an expert system can help us gain more insights on the origin of **backward** and **forward** chaining in artificial intelligence.
- An expert system is a computer application that uses rules, approaches, and facts to provide solutions to complex problems.
- Examples of expert systems include **MYCIN** and **DENDRAL**.
- MYCIN uses the backward chaining technique to diagnose bacterial infections.
- DENDRAL employs forward chaining to establish the structure of chemicals.

Expert system and Chaining: Relationship

- There are **three** components in an expert system: **user interface, inference engine**, and **knowledge base**.
- The user interface enables users of the system to interact with the expert system. High-quality and domain-specific knowledge is stored in the **knowledge base**.

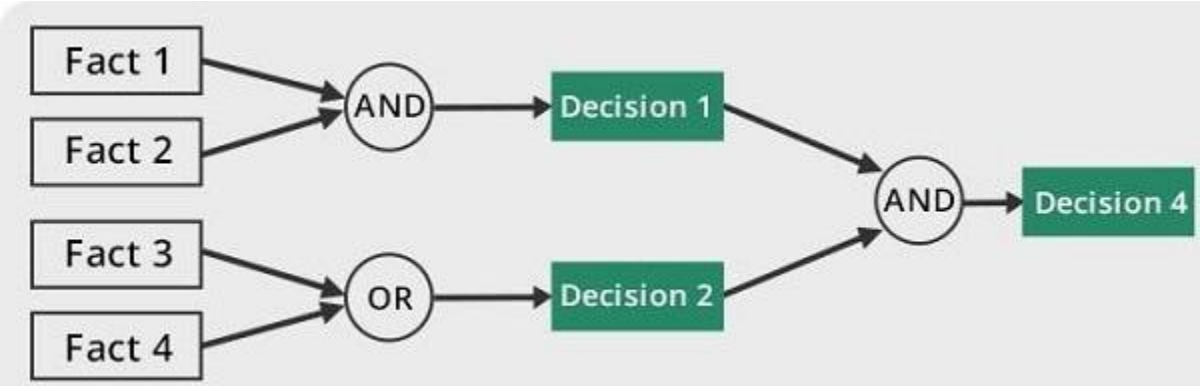


- Backward and forward chaining stem from the **inference engine** component.
- This is a component in which **logical rules** are applied to the knowledge base to get new information or make a decision.
- The backward and forward chaining techniques are used by the inference engine as strategies for **proposing solutions** or **deducing information** in the expert system.

Forward chaining / Forward reasoning

- Forward chaining is a method of reasoning in artificial intelligence in which inference rules are applied to existing data to extract additional data until an endpoint (goal) is achieved.
- In this type of chaining, the inference engine starts by evaluating existing facts, derivations, and conditions before deducing new information.
- An endpoint (goal) is achieved through the manipulation of knowledge that exists in the knowledge base.

"What can happen next?"



Properties of forward chaining

- The process uses a down-up approach (bottom to top).
- It starts from an initial state and uses facts to make a conclusion.
- This approach is data-driven.
- It's employed in expert systems and production rule system.

Examples of forward chaining

A simple example of forward chaining can be explained in the following sequence.

A
A → B
B

A is the starting point. A → B represents a fact. This fact is used to achieve a decision B. **(Modes Ponens)**

A practical example will go as follows;

Tom is running (A)
If a person is running, he will sweat (A → B)
Therefore, Tom is sweating. (B)

Advantages & Disadvantages: Forward Chaining

Advantages

- It can be used to draw multiple conclusions.
- It provides a good basis for arriving at conclusions.
- It's more flexible than backward chaining because it does not have a limitation on the data derived from it.

Disadvantages

- The process of forward chaining may be time-consuming.
- It may take a lot of time to eliminate and synchronize available data.
- Unlike backward chaining, the explanation of facts or observations for this type of chaining is not very clear.

Forward Chaining Example: I

suppose these are the things we have in our knowledge base:

- P implies Q
- L and M implies P
- B and L implies M
- A and P implies L
- A and B implies L
- A is true
- B is true.
- And now I asked the question is Q true?

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

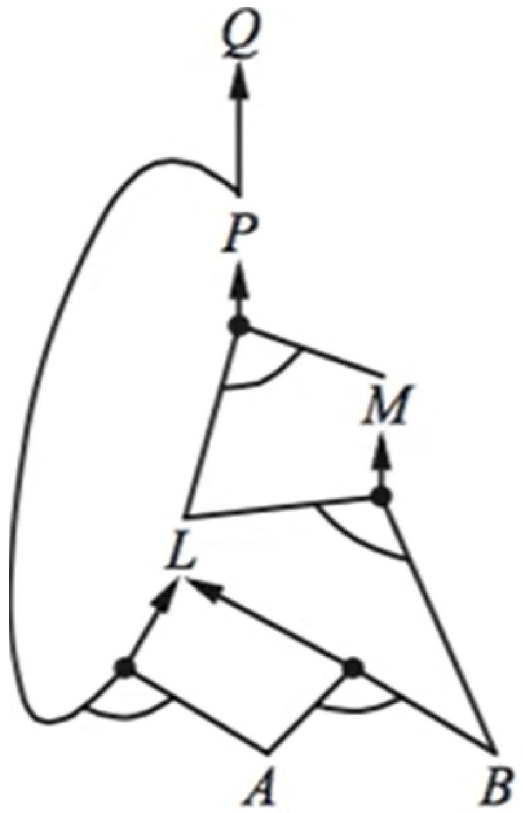
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

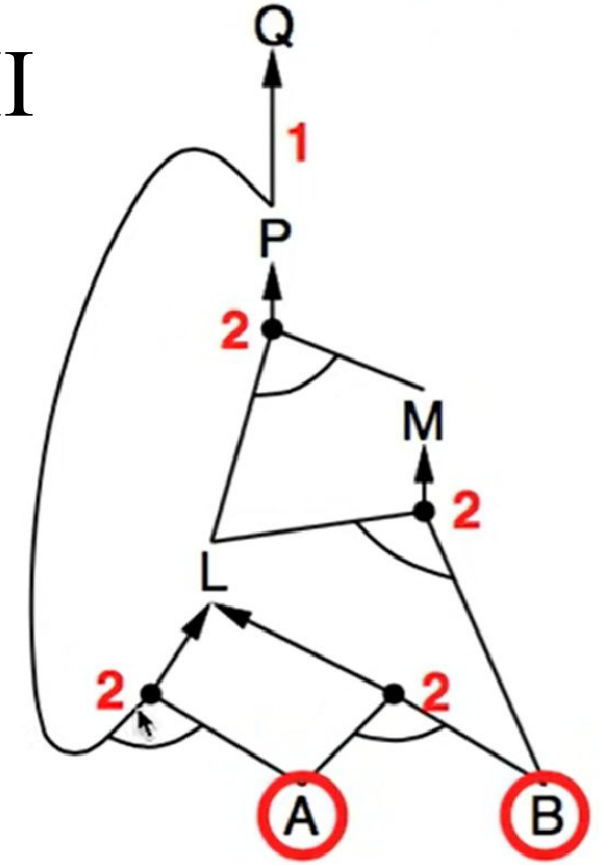
(a)



(b)

Forward Chaining Example: II

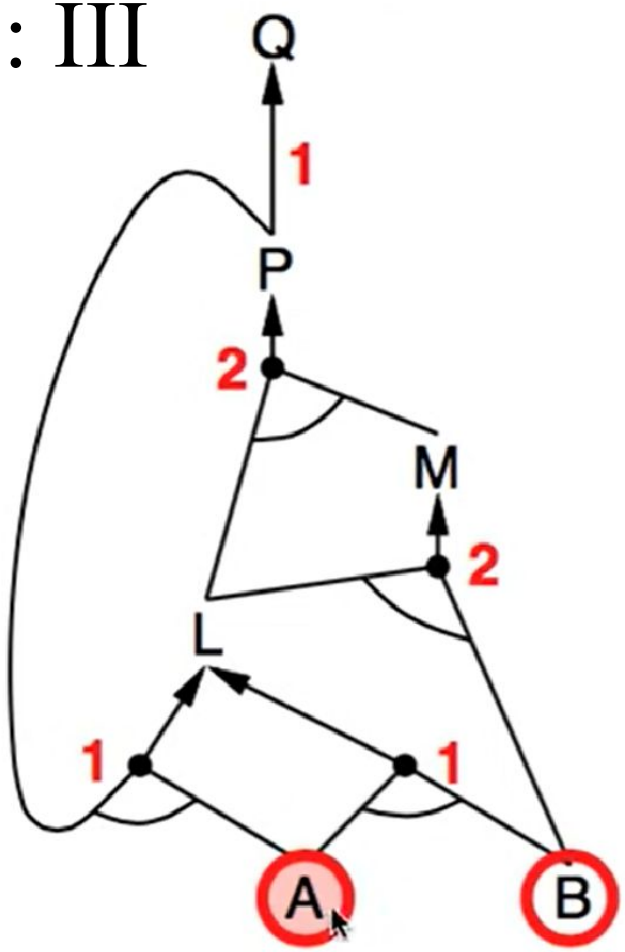
- Q has only one literal
- L has 2 literals A and P
- M has 2 literals B and L
- And so on…



$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

- And I put A and B in my queue, because those are the 2 things I know to be true.
- Now, I can run a simple forward chaining algorithm. Because A is true I pop it out from the queue.

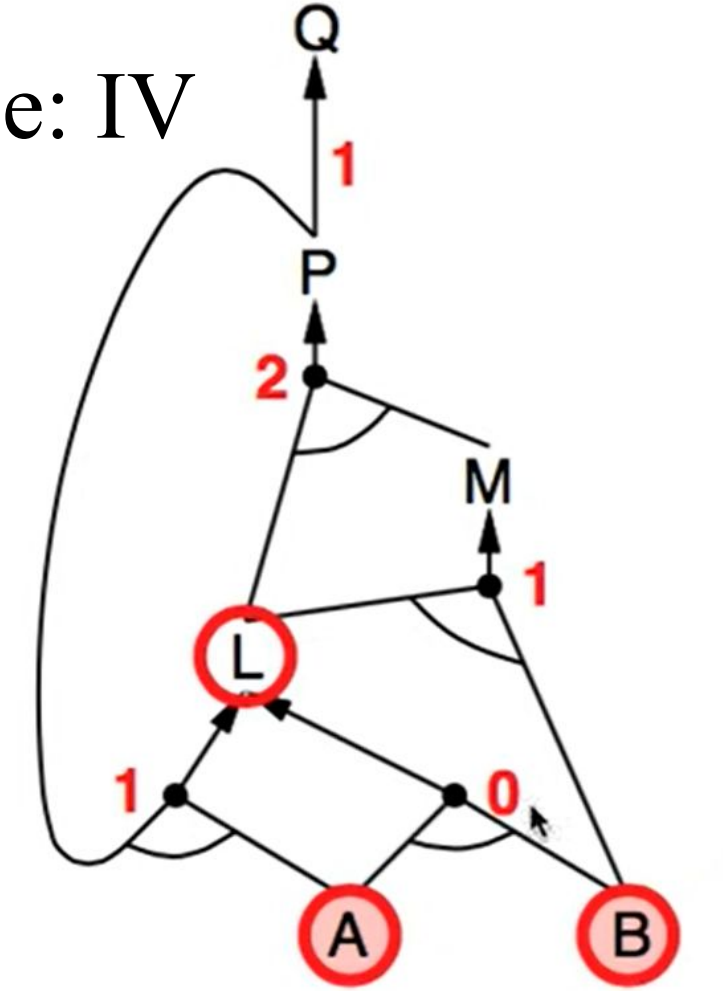
Forward Chaining Example: III



$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

- As soon as I know A is true, I know that the dot for A and B (the node for A and B), no longer needs 2 things to be true it only needs one more thing to be true.

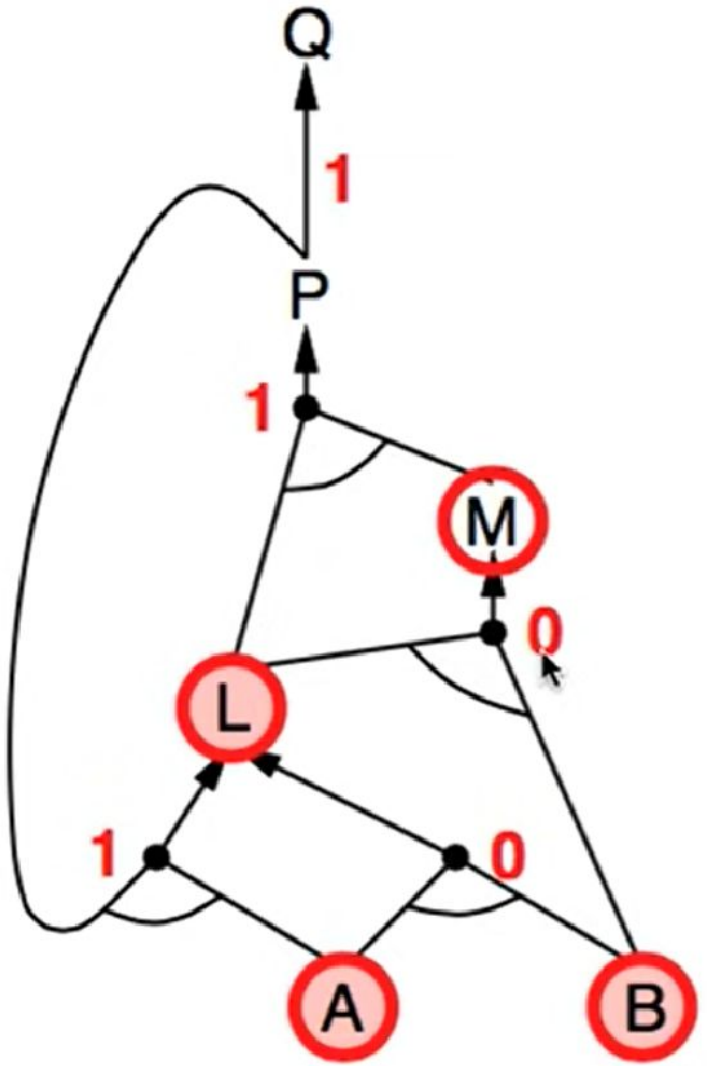
Forward Chaining Example: IV



$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

- As soon as I pop B, now I know that I do not need to satisfy both the things of this antecedent are satisfied.
- Because both the things of this antecedent are satisfied now I can prove L because I can prove L, I put it in the queue and I repeat this process.

Forward Chaining Example: V



$P \Rightarrow Q$

- $L \wedge M \Rightarrow P$
- $B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

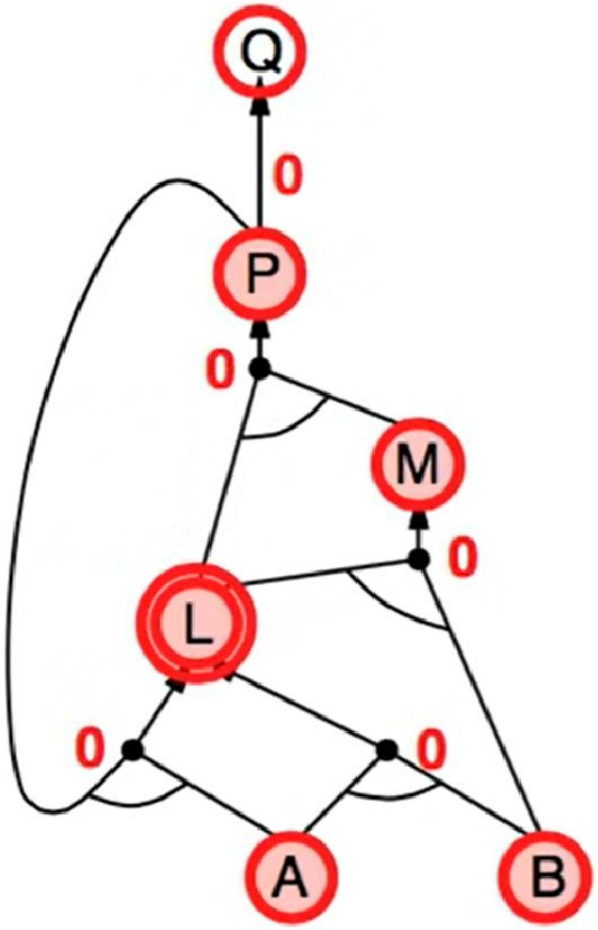
$A \wedge B \Rightarrow L$

A

B

Forward Chaining Example: VI

- And suppose, I keep running this procedure, and at some point, I get to Q to be true and when I can get Q to be true, that means I have solved the question I had at hand.
- So now you see that this is an inference procedure, which is using a mechanical process for computing new sentences.
- And in this case, if all the sentences are horn clauses (a clause with at most one positive literal) of the form, A and B and C and so, implies something and my query is a single variable, or a conjunction of multiple variables, but not a disjunction.
- So, if my query is a single variable, then I can say that this procedure is guaranteed to prove it and it will be also complete.



$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Example: Forward Chaining

Suppose that the goal is to conclude the **color of a pet named Fritz**, given that **he croaks** and **eats flies**, and that the rule base contains the following four rules:

1. If X croaks and X eats flies - Then X is a frog
2. If X chirps and X sings - Then X is a canary
3. If X is a frog - Then X is green
4. If X is a canary - Then X is yellow

Let us illustrate forward chaining by following the pattern of a computer as it evaluates the rules.

Assume the following facts:

Fritz croaks

Fritz eats flies

With forward reasoning, the inference engine can derive that **Fritz is green** in a series of steps:

1. Since the base facts indicate that "Fritz croaks" and "Fritz eats flies", the antecedent of rule #1 is satisfied by substituting Fritz for X, and the inference engine concludes:

Fritz is a frog

2. The antecedent of rule #3 is then satisfied by substituting Fritz for X, and the inference engine concludes:

Fritz is green

- **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.**

- We first represent these facts in first-order definite clauses.

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

   $American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \rightarrow Criminal(x)$

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

2. "Nono ... has some missiles": $\exists x\ Owns(Nono, x) \land Missile(x)$ is transformed into two definite clauses by Existential Instantiation, introducing a new constant M_1:

   Owns(Nono, M1)
   Missile(M1)

RES: *Owns(Nono, M1)*
Missile(M1)

- All of the missiles were sold to country Nono by Colonel West.

  $\forall$ x Missiles(x) $\wedge$ Owns (Nono, x) $\rightarrow$ Sells (West, x, Nono)

RES: ***¬Missiles(x) ∨ ¬Owns (Nono, x) ∨ Sells (West, x, Nono)***

- We will also need to know that "missiles are weapons".

  Missile(x) $\rightarrow$ Weapons (x)

RES: ***¬ Missile(x) ∨ Weapons (x)***

- We know that enemy of America counts as "hostile".
  Enemy(x, America) $\rightarrow$ Hostile(x)

RES: ***¬Enemy(x, America) ∨ Hostile(x)***

- West, who is American …
  American(West) **RES:** ***American(West)***

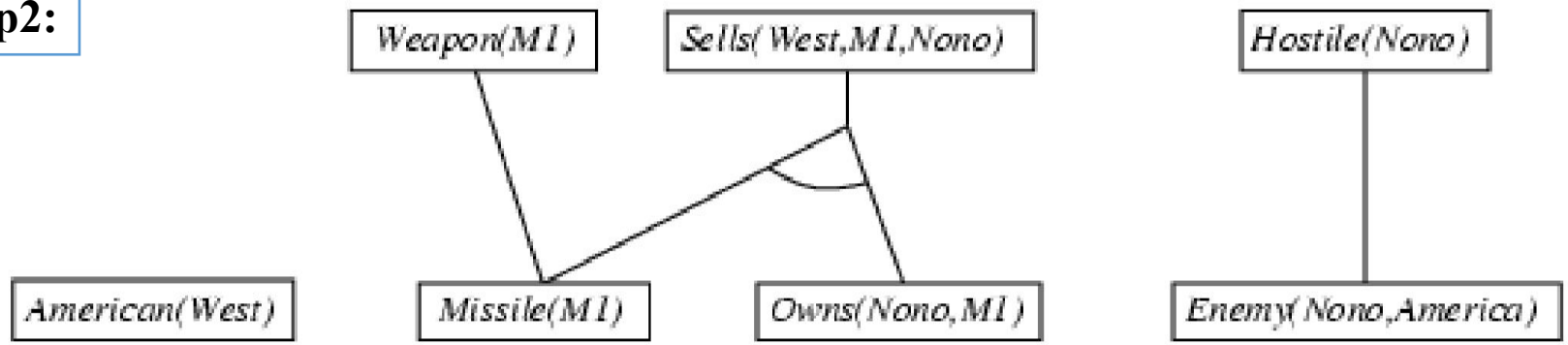
- The country Nono, an enemy of America …

  Enemy (Nono, America) **RES:** ***Enemy (Nono, America)***

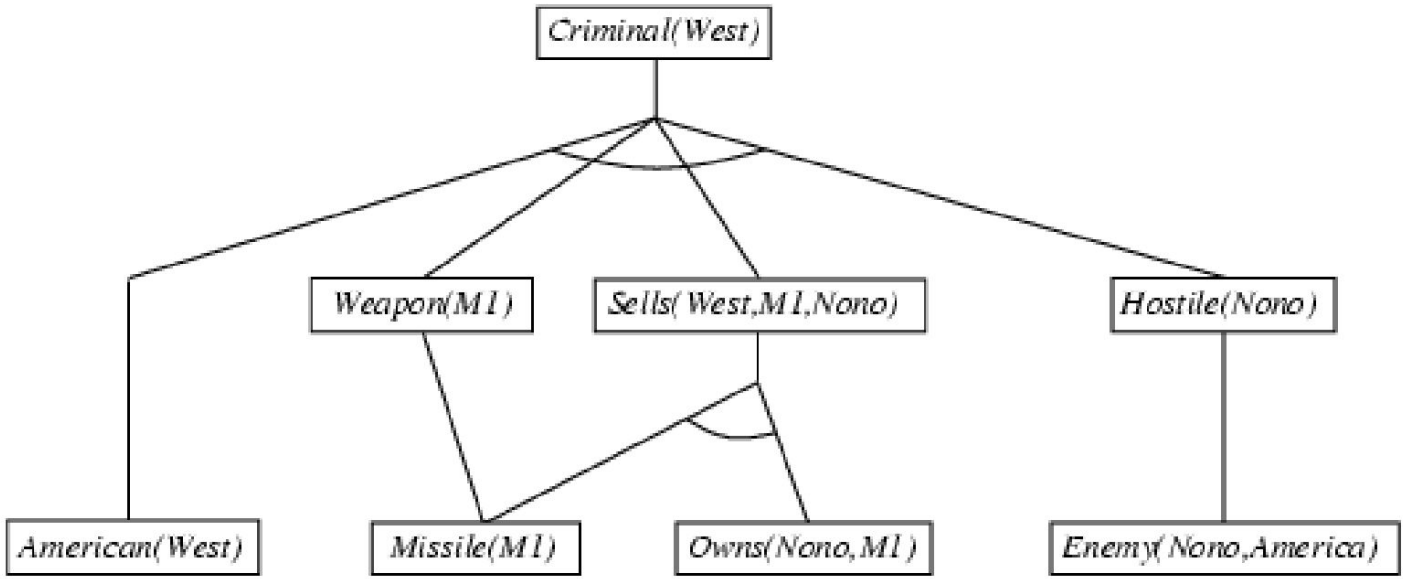
Step1:

American(West) Missile(M1) Owns(Nono,M1) Enemy(Nono,America)

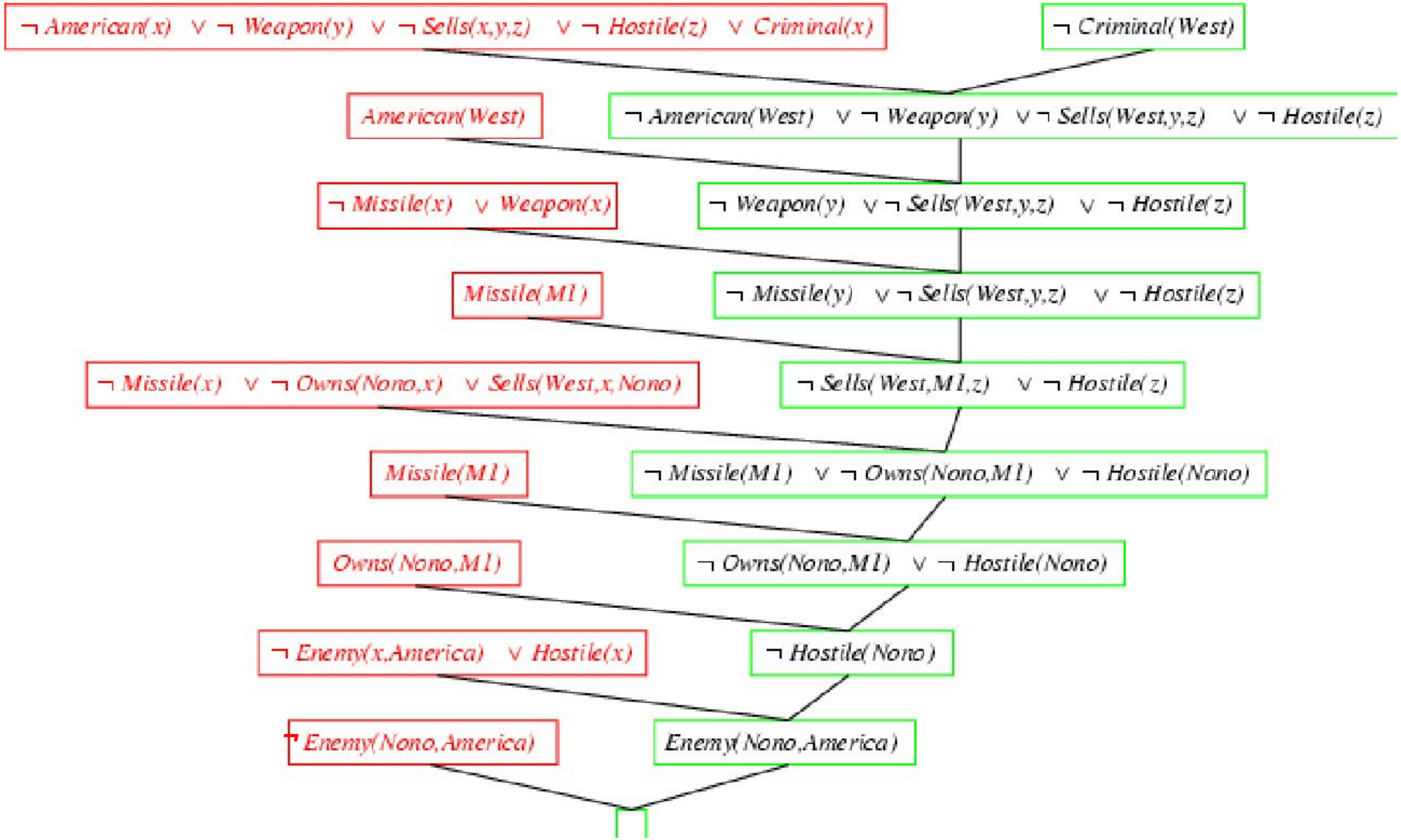
Step2:



Step3: We can check the given statement that needs to be checked and check whether it is satisfied with the substitution which infers all the previously stated facts. Thus we reach our goal.



Resolution proof: West is a criminal (previous example)



Backward Chaining

- Backward chaining is a concept in artificial intelligence that involves backtracking from the endpoint or goal to steps that led to the endpoint.
- This type of chaining starts from the goal and moves backward to comprehend the steps that were taken to attain this goal.
- The backtracking process can also enable a person establish logical steps that can be used to find other important solutions.

“Why this happened?”