Knowledge Representation

Prof. Deepak C Vegda
email.:deepakvegda.it@ddu.ac.in

What is knowledge representation?

- Humans are best at understanding, reasoning, and interpreting knowledge. Human knows things, which is knowledge and as per their knowledge they perform various actions in the real world. **But how machines do all these things comes under knowledge representation and reasoning**.

What is knowledge representation?

- Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which concerned with AI agents thinking and how thinking contributes to intelligent behavior of agents..

What is knowledge representation?

- It is responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve the complex real world problems such as diagnosis a medical condition or communicating with humans in natural language.

What is knowledge representation?

- It is also a way which describes how we can represent knowledge in artificial intelligence. Knowledge representation is not just storing data into some database, but it also enables an intelligent machine to learn from that knowledge and experiences so that it can behave intelligently like a human.

What to Represent:

- **Object:** All the facts about objects in our world domain. E.g., Guitars contains strings, trumpets are brass instruments.
- **Events:** Events are the actions which occur in our world.
- **Performance:** It describe behaviour which involves knowledge about how to do things.
- **Meta-knowledge:** It is knowledge about what we know.
- **Facts:** Facts are the truths about the real world and what we represent.
- **Knowledge-Base:** The central component of the knowledge-based agents is the knowledge base. It is represented as KB. The Knowledgebase is a group of the Sentences (Here, sentences are used as a technical term and not identical with the English language).

Types of Knowledge

1. **Declarative Knowledge**
2. **Procedural Knowledge**
3. **Meta-knowledge**
4. **Heuristic knowledge**
5. **Structural knowledge**

Declarative knowledge is to know about something.
It includes concepts, facts, and objects.
It is also called descriptive knowledge and expressed in declarative sentences.
It is simpler than procedural language.

Procedural Knowledge

- It is also known as imperative knowledge.
- Procedural knowledge is a type of knowledge which is responsible for knowing how to do something.
- It can be directly applied to any task.
- It includes rules, strategies, procedures, agendas, etc.
- Procedural knowledge depends on the task on which it can be applied

Meta-knowledge

- Knowledge about the other types of knowledge is called Meta-knowledge.

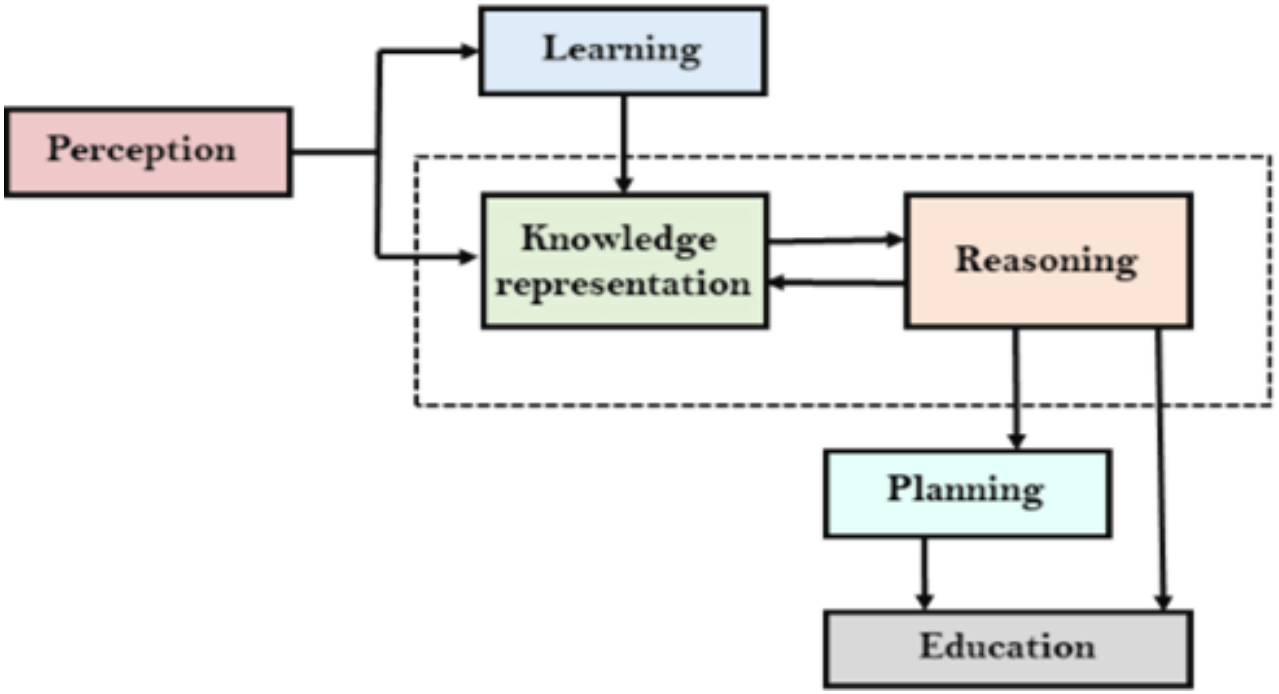
Heuristic knowledge

- Heuristic knowledge is representing knowledge of some experts in a filed or subject.
- Heuristic knowledge is rules of thumb based on previous experiences, awareness of approaches, and which are good to work but not guaranteed.

Structural knowledge

- Structural knowledge is basic knowledge to problem-solving.
- It describes relationships between various concepts such as kind of, part of, and grouping of something.
- It describes the relationship that exists between concepts or objects

AI Knowledge Cycle



Requirements for knowledge Representation system:

1. **Representational Accuracy:**
   KR system should have the ability to represent all kind of required knowledge.
2. **Inferential Adequacy:**
   KR system should have ability to manipulate the representational structures to produce new knowledge corresponding to existing structure.
3. **Inferential Efficiency:**
   The ability to direct the inferential knowledge mechanism into the most productive directions by storing appropriate guides.
4. **Acquisitional efficiency-** The ability to acquire the new knowledge easily using automatic methods.

Techniques of knowledge representation

- Logical Representation
- Semantic Network Representation
- Frame Representation
- Production Rule

Logical Representation

- Logical representation is a language with some concrete rules which deals with propositions and has no ambiguity in representation.
- It consists of precisely defined syntax and semantics which supports the sound inference.

Syntax

Syntaxes are the rules which decide how we can construct legal sentences in the logic.

It determines which symbol we can use in knowledge representation.

How to write those symbols.

Semantics

- Semantics are the rules by which we can interpret the sentence in the logic.
- Semantic also involves assigning a meaning to each sentence.

Logical Representation

- Propositional Logics
- Predicate logics

Logical Representation

- Advantages of logical representation:
  - Logical representation enables us to do logical reasoning.
  - Logical representation is the basis for the programming languages.
- Disadvantages of logical Representation:
  - Logical representations have some restrictions and are challenging to work with.
  - Logical representation technique may not be very natural, and inference may not be so efficient.

Propositional Logic

- Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions. A proposition is a declarative statement which is either true or false. It is a technique of knowledge representation in logical and mathematical form

Propositional Logic

- Propositional logic is also called Boolean logic as it works on 0 and 1.
- In propositional logic, we use symbolic variables to represent the logic
- Propositions can be either true or false, but it cannot be both.
- Propositional logic consists of an object, relations or function, and **logical connectives**
- These connectives are also called logical operators
- The propositions and connectives are the basic elements of the propositional logic.
- Connectives can be said as a logical operator which connects two sentences.

Propositional Logic

- A proposition formula which is always true is called **tautology**, and it is also called a valid sentence
- A proposition formula which is always false is called **Contradiction**.
- Statements which are questions, commands, or opinions are not propositions such as "**Where is Akshay**", "**How are you**", "**What is your name**", are not propositions.

Atomic Propositions & Compound propositions

- Atomic propositions are the simple propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false.

  "The Sun is cold“

- Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.

  "It is raining today, and street is wet."

Logical Connectives

- **Negation:** A sentence such as ¬ P is called negation of P. A literal can be either Positive literal or negative literal.
- **Conjunction:** A sentence which has ∧ connective such as, **P ∧ Q** is called a conjunction.
  Example: Rohan is intelligent and hardworking. It can be written as,
  P= Rohan is intelligent,
  Q= Rohan is hardworking. → P∧ Q.
- **Disjunction:** A sentence which has ∨ connective, such as **P ∨ Q**. is called disjunction, where P and Q are the propositions.
  Example: "Ritika is a doctor or Engineer",
  Here P= Ritika is Doctor. Q= Ritika is Doctor, so we can write it as **P ∨ Q**.
- **Implication:** A sentence such as P → Q, is called an implication. Implications are also known as if-then rules. It can be represented as
  If it is raining, then the street is wet.
  Let P= It is raining, and Q= Street is wet, so it is represented as P → Q
- **Biconditional:** A sentence such as **P⇔ Q is a Biconditional sentence, example If I am breathing, then I am alive**
  P= I am breathing, Q= I am alive, it can be represented as P ⇔ Q.

Connective symbols	Word	Technical term	Example
$\wedge$	AND	Conjunction	$A \wedge B$
$\vee$	OR	Disjunction	$A \vee B$
$\rightarrow$	Implies	Implication	$A \rightarrow B$
$\Leftrightarrow$	If and only if	Biconditional	$A \Leftrightarrow B$
$\neg$ or ~	Not	Negation	$\neg A$ or $\neg B$

Truth Table for connectives

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

P	Q	R	¬R	P v Q	P v Q → ¬R
True	True	True	False	True	False
True	True	False	True	True	True
True	False	True	False	True	False
True	False	False	True	True	True
False	True	True	False	True	False
False	True	False	True	True	True
False	False	True	False	False	True
False	False	False	True	False	True

Precedence of the operators

Precedence	Operators
First Precedence	Parenthesis
Second Precedence	Negation
Third Precedence	Conjunction(AND)
Fourth Precedence	Disjunction(OR)
Fifth Precedence	Implication
Six Precedence	Biconditional

Properties of Operators

- **Commutativity:**
  - $P \wedge Q = Q \wedge P$, or
  - $P \vee Q = Q \vee P$.
- **Associativity:**
  - $(P \wedge Q) \wedge R = P \wedge (Q \wedge R)$,
  - $(P \vee Q) \vee R = P \vee (Q \vee R)$
- **Identity element:**
  - $P \wedge \text{True} = P$,
  - $P \vee \text{True} = \text{True}$.
- **Distributive:**
  - $P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$.
  - $P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$.
- **DE Morgan's Law:**
  - $\neg (P \wedge Q) = (\neg P) \vee (\neg Q)$
  - $\neg (P \vee Q) = (\neg P) \wedge (\neg Q)$.
- **Double-negation elimination:**
  - $\neg (\neg P) = P$.

Logical equivalence

Two sentences are logically equivalent iff true in same models:

$\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha) \quad \text{commutativity of } \wedge$$

$$(\alpha \vee \beta) \equiv (\beta \vee \alpha) \quad \text{commutativity of } \vee$$

$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma)) \quad \text{associativity of } \wedge$$

$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma)) \quad \text{associativity of } \vee$$

$$\neg(\neg\alpha) \equiv \alpha \quad \text{double-negation elimination}$$

$$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{contraposition}$$

$$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta) \quad \text{implication elimination}$$

$$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination}$$

$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \quad \text{De Morgan}$$

$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \quad \text{De Morgan}$$

$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributivity of } \wedge \text{ over } \vee$$

$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributivity of } \vee \text{ over } \wedge$$

Validity and satisfiability

A sentence is valid if it is true in *all* models,

e.g., true, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$

Validity is connected to inference via the Deduction Theorem:

$KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid

A sentence is satisfiable if it is true in *some* model

e.g., $A \vee B$, C

A sentence is unsatisfiable if it is true in *no* models

e.g., $A \wedge \neg A$

Satisfiability is connected to inference via the following:

$KB \models \alpha$ if and only if $(KB \wedge \neg\alpha)$ is unsatisfiable

i.e., prove α by *reductio ad absurdum*

examples

$(p \rightarrow q) \wedge (q \rightarrow p)$

$\neg(p \vee \neg(p \wedge q))$

$(p \rightarrow q) \wedge \neg q \Rightarrow \neg p$

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

$[(p \rightarrow q) \wedge (q \rightarrow r)] \Rightarrow (p \rightarrow r)$

Rules of Inference in Artificial intelligence

- **Implication:** It is one of the logical connectives which can be represented as P → Q. It is a Boolean expression.
- **Converse:** The converse of implication, which means the right-hand side proposition goes to the left-hand side and vice-versa. It can be written as Q → P.
- **Contrapositive:** The negation of converse is termed as contrapositive, and it can be represented as ¬ Q → ¬ P.
- **Inverse:** The negation of implication is called inverse. It can be represented as ¬ P → ¬ Q.

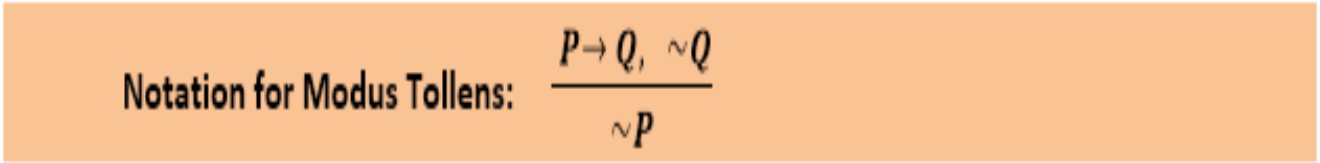
Types of Inference rules

1. Modus Ponens:
   - The Modus Ponens rule is one of the most important rules of inference, and it states that if P and P → Q is true, then we can infer that Q will be true. It can be represented as:

Notation for Modus ponens: $\frac{P \rightarrow Q, \quad P}{\therefore Q}$

Types of Inference rules

2. Modus Tollens:
   - The Modus Tollens rule state that if P→ Q is true and **¬ Q is true, then ¬ P** will also true. It can be represented as



Types of Inference rules

3. Hypothetical Syllogism:
   - The Hypothetical Syllogism rule state that if P→R is true whenever P→Q is true, and Q→R is true. It can be represented as the following notation:

Example:

   - **Statement-1:** If you have my home key then you can unlock my home. **P→Q**
     Statement-2: If you can unlock my home then you can take my money. **Q→R**
     Conclusion: If you have my home key then you can take my money. **P→R**

Types of Inference rules

4. Disjunctive Syllogism:
   - The Disjunctive syllogism rule state that if P∨Q is true, and ¬P is true, then Q will be true. It can be represented as:

Notation of Disjunctive syllogism: $\frac{P \vee Q,\ \neg P}{Q}$

Types of Inference rules

5. Resolution:
   - The Resolution rule state that if P∨Q and ¬ P∧R is true, then Q∨R will also be true. **It can be represented as**

Notation of Resolution $\frac{P \vee Q, \ \neg P \wedge R}{Q \vee R}$

Types Applications of Propositional Logic : CSP

You are planning a party, but your friends are a bit touchy about who will be there.

1. If John comes, he will get very hostile if Sarah is there.
2. Sarah will only come if Kim will be there also.
3. Kim says she will not come unless John does.

- Who can you invite without making someone unhappy?

Translate the information

J : John Comes to party

K : Kim comes to party

S : Sarah comes to party

Represent

1. If John comes, he will get very hostile if Sarah is there.
2. Sarah will only come if Kim will be there also.
3. Kim says she will not come unless John does.

1

2

3

Represent

1. If John comes, he will get very hostile if Sarah is there.
2. Sarah will only come if Kim will be there also.
3. Kim says she will not come unless John does.

1 $J \rightarrow \neg S$

2 $S \rightarrow K$

3 $K \rightarrow J$

Thus, for a successful party to be possible, we want the formula $(J \rightarrow \neg S) \wedge (S \rightarrow K) \wedge (K \rightarrow J)$ to be satisfiable.

Knight Knave Puzzle

On a mystical island, there are two kinds of people: knights and knaves. Knights always tell the truth. Knaves always lie.

You meet two people on the island, A and B.

A says “We are both knights”

B says “A is lying!”

- A: Arnold is a knight
- B: Bob is a knight

Truth Table

A	B	A^B	~A	A<->(A^B)	B<->~A	1 ^2
T	T	T	F	F	T	
F	T	F	T	T	T	T
T	F	F	F	F	F	
F	F	F	T	T	T	

Example

The propositional logic has very limited expressive power;

Some humans are intelligent.

All man are mortal.

Marcus is man.

First-Order logic

- An extension of PL.
- Sufficiently expressive to represent natural language concisely.
- It is also known as **Predicate logic or First-order predicate logic**

First-Order logic

- It is not only about facts;
  - **Objects:** A, B, people, numbers, colors, wars, theories, squares....
  - **Relations: It can be unary relation such as:** red, round, is adjacent, **or n-any relation such as:** the sister of, brother of, has color, comes between
  - **Function:** Father of, best friend, third inning of, end of, ……

FOL

First-order logic statements can be divided into two parts:

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

Ex. Marcus was a roman.

Quantifiers in First-order logic

- **Universal Quantifier, (for all, everyone, everything)** **∀x**
- **Existential quantifier, (for some, at least one)** ∃x

In universal quantifier we use implication "→".

In Existential quantifier we always use AND or Conjunction symbol (∧).

Example

“All blocks are red”

“There is a block A”

The alphabets in FOL

Symbols:

- Operators: ¬, ∨, ∧, ∀, ∃, =
- Variables: x, x1, x2, . . . , y, . . . , z, . . .
- Brackets: (), [], {}
- Function symbols (e.g., weight(), color ())
- Predicate symbols (e.g., block(), red())
- Predicate and function symbols have an arity (number of arguments).

Examples

- Every gardener likes the sun
- All purple mushrooms are poisonous
- **John likes all kind of food.**
- **John likes all kind of food.**

Lets Try….

(a) Marcus was a man.

(b) Marcus was a Roman.

(c) All men are people.

(d) Caesar was a ruler.

(e) All Romans were either loyal to Caesar or hated him (or both).

(f) Everyone is loyal to someone.

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

(h) Marcus tried to assassinate Caesar.

(a) Marcus was a man.

(b) Marcus was a Roman.

(c) All men are people.

(d) Caesar was a ruler.

(e) All Romans were either loyal to Caesar or hated him (or both).

(f) Everyone is loyal to someone.

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

(h) Marcus tried to assassinate Caesar.

(a) $man(marcus)$

(b) $roman(marcus)$

(c) $\forall X.\ man(X) \rightarrow person(X)$

(d) $ruler(caesar)$

(e) $\forall X.\ roman(x) \rightarrow loyal(X,caesar) \vee hate(X,caesar)$

(f) $\forall X \exists Y.\ loyal(X,Y)$

(g) $\forall X \forall Y.\ person(X) \wedge ruler(Y) \wedge tryassasin(X,Y) \rightarrow \neg loyal(X,Y)$

(h) $tryassasin(marcus,caesar)$

Example

1. All birds fly.
2. Every man respects his parent.
3. Some boys play cricket.
4. Not all students like both Mathematics and Science.
5. Only one student failed in Mathematics.

Resolution in FOL

- Resolution is used, if there are various statements are given, and we need to prove a conclusion of those statements.
- Unification is a key concept in proofs by resolutions. Resolution is a single inference rule which can efficiently operate on the **conjunctive normal form or clausal form**.

Steps for Resolution

1. Conversion of facts into first-order logic.
2. Convert FOL statements into CNF
3. Negate the statement which needs to prove (proof by contradiction)
4. Draw resolution graph (unification).

Resolution

1. **John likes all kind of food.**
2. **Apple and vegetable are food**
3. **Anything anyone eats and not killed is food.**
4. **Anil eats peanuts and still alive**
5. **Harry eats everything that Anil eats.**
   Prove by resolution that:
6. **John likes peanuts.**

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)

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

} **added predicates.**

h. likes(John, Peanuts)

Covert in form of CNF

- Eliminate all implications →
- Reduce the scope of all ¬ to single term
- Make all variable names unique
- Move quantifiers left (prenex normal form)
- Eliminate Existential Quantifiers
- Eliminate Universal Quantifiers
- Convert to conjunction of disjuncts
- Create separate clause for each conjunct.

Step-2: Conversion of FOL into CNF

- **Eliminate all implication (→) and rewrite**
  - ∀x ¬ food(x) V likes(John, x)
  - food(Apple) Λ food(vegetables)
  - ∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)
  - eats (Anil, Peanuts) Λ alive(Anil)
  - ∀x ¬ eats(Anil, x) V eats(Harry, x)
  - ∀x¬ [¬ killed(x)] V alive(x)
  - ∀x ¬ alive(x) V ¬ killed(x)
  - likes(John, Peanuts).

- **Move negation (¬)inwards and rewrite**
  - ∀x ¬ food(x) V likes(John, x)
  - food(Apple) Λ food(vegetables)
  - ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)
  - eats (Anil, Peanuts) Λ alive(Anil)
  - ∀x ¬ eats(Anil, x) V eats(Harry, x)
  - ∀x ¬killed(x)] V alive(x)
  - ∀x ¬ alive(x) V ¬ killed(x)
  - likes(John, Peanuts).

- **Rename variables or standardize variables**
  - ∀x ¬ food(x) V likes(John, x)
  - food(Apple) Λ food(vegetables)
  - ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)
  - eats (Anil, Peanuts) Λ alive(Anil)
  - ∀w¬ eats(Anil, w) V eats(Harry, w)
  - ∀g ¬killed(g)] V alive(g)
  - ∀k ¬ alive(k) V ¬ killed(k)
  - likes(John, Peanuts).

- **Eliminate existential instantiation quantifier by elimination.**
  In this step, we will eliminate existential quantifier ∃, and this process is known as **Skolemization**. But in this example problem since there is no existential quantifier so all the statements will remain same in this step.

Skolemization Example

- In general the function must have the same number of arguments as the number of universal quantifiers in the current scope.
- Example: ∀x ∃y Father (y, x)

  – create a new function named foo and replace y with the function.

  – ∀x Father (foo(x), x)

Drop Universal quantifiers

In this step we will drop all universal quantifier since all the statements are not implicitly quantified so we don't need it.¬ food(x) V likes(John, x)

- food(Apple)
- food(vegetables)
- ¬ eats(y, z) V killed(y) V food(z)
- eats (Anil, Peanuts)
- alive(Anil)
- ¬ eats(Anil, w) V eats(Harry, w)
- killed(g) V alive(g)
- ¬ alive(k) V ¬ killed(k)
- likes(John, Peanuts).

Unification

- Two formulas are said to unify if there are legal instantiations (assignments of terms to variables) that make the formulas in question identical.
- The act of unifying is called unification. The instantiation that unifies the formulas in question is called a unifier.
- There is a simple algorithm called the unification algorithm that does this.

Unification

- **Example**: Unify the formulas *Q(a, y, z)* and *Q(y, b, c)*
- **Solution**:
  - Since *y* in *Q(a, y, z)* is a different variable than *y* in *Q(y, b, c),* rename *y* in the second formula to become *y1*.
  - This means that one must unify *Q(a, y, z)* with *Q(y1, b, c).*
  - An instance of *Q(a, y, z)* is *Q(a, b, c)* and an instance of *Q(y1, b, c)* is *Q(a, b, c).*
  - Since these two instances are identical, *Q(a, y, z)* and *Q(y, b, c)* unify.
  - The unifier is *y1 = a, y = b, z = c.*

Unification

- **Unification**: matching literals and doing substitutions that resolution can be applied.
- **Substitution**: when a variable name is replaced by another variable or element of the domain.
  - Notation [a/x] means replacing all occurrences of x with a in the formula
  - Example: substitution [5/x] in p(x) ∨ Q(x,y) results in p(5) ∨ Q(5,y)

Convert to CNF Algo

(1) Eliminate conditionals $\rightarrow$, using the equivalence

$p \rightarrow q \equiv \neg p \vee q$

e.g. $(\exists x)(p(x) \wedge (\forall y)(f(y) \rightarrow h(x,y)))$ becomes

$(\exists x)(p(x) \wedge (\forall y)(\neg f(y) \vee h(x,y)))$

(2) Eliminate negations or reduce the scope of negation to one atom.

e.g. $\neg\neg p \equiv p$

$\neg(p \wedge q) \equiv \neg p \vee \neg q$

$\neg(\forall x \in S, F(x)) \equiv \exists x \in S,\ \neg F(x)$

$\neg(\exists x \in S, F(x)) \equiv \forall x \in S,\ \neg F(x)$

(3) Standardize variables within a well-formed formula so that the bound or free variables of each quantifier have unique names. e.g. $(\exists x)\neg p(x) \vee (\forall x)p(x)$ is replaced by $(\exists x)\neg p(x) \vee (\forall y)p(y)$

Convert to CNF Algo

(4) Advanced step: if there are existential quantifiers, eliminate them by using Skolem functions

e.g. $(\exists x)p(x)$ is replaced by $p(a)$

$(\forall x)(\exists y)k(x,y)$ is replaced by $(\forall x)\ k(x,f(x))$

(5) Convert the formula to prenex form

e.g. $(\exists x)(p(x) \wedge (\forall y)\ (\neg f(y) \vee h(x,y)))$ becomes

$(\forall y)\ (p(a) \wedge (\neg f(y) \vee h(a,y)))$

(6) Convert the formulas to CNF, which is a conjunctive of clauses. Each clause is a disjunction.

e.g. $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$

(7) Drop the universal quantifiers

e.g. the formula in (5) becomes $p(a) \wedge (\neg f(y) \vee h(a,y))$

Convert to CNF Algo

(8) Eliminate the conjunctive signs by writing the formula as a set of clauses

e.g. $p(a) \wedge (\neg f(y) \vee h(a,y))$ becomes $p(a)$, $(\neg f(y) \vee h(a,y))$

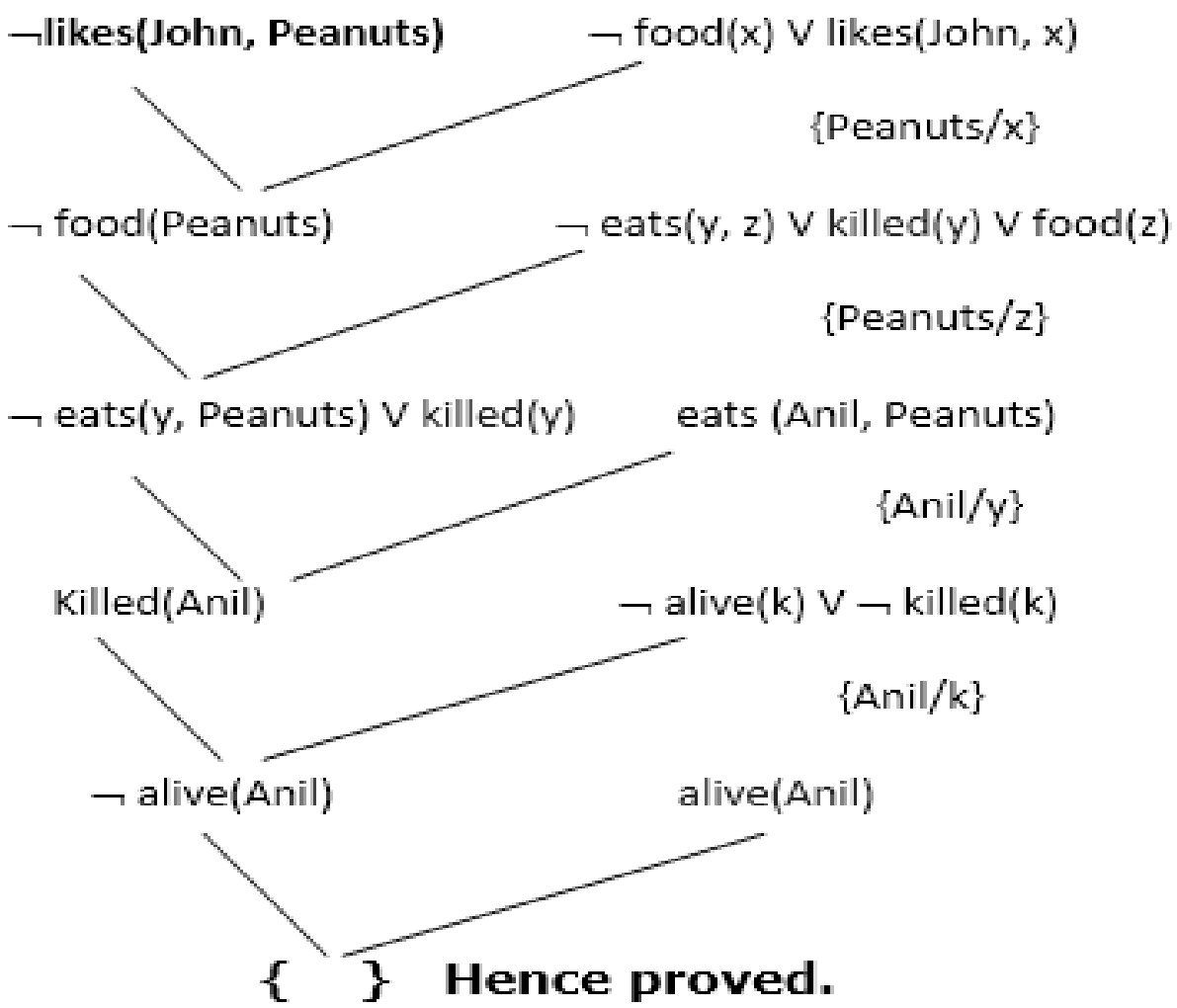
(9) Rename variables in clauses, if necessary, so that the same variable name is only used in one clause.

e.g. $p(x) \vee q(y) \vee k(x,y)$ and $\neg p(x) \vee q(y)$ becomes

$p(x) \vee q(y) \vee k(x,y)$ and $\neg p(x1) \vee q(y1)$

Negate the statement to be proved

- In this statement, we will apply negation to the conclusion statements, which will be written as ¬likes(John, Peanuts)


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

1. man (marcus)

2. pompeian (marcus)

3. $\forall x$: pompeian (x) $\rightarrow$ roman(x)

   $\therefore$ $\neg$ pompeian (x) $\vee$ roman x)

4. ruler (caesar)

5. $\forall x$: roman (x) $\rightarrow$ loyal to (x, caesar) $\vee$ hated (x, caesar)

   $\therefore$ $\neg$ roman (p) $\vee$ loyal to (p, caesar) $\vee$ hated (p, caesar)

6. $\forall x \cdot \exists y$ loyal to (x, y)

   $\therefore$ loyal to (a, f(a))

7. $\forall x \forall y$ people (x) $\wedge$ try to assassinate (x, y) $\wedge$ ruler (y).

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

$\therefore$ $\neg$ people (x_1) $\vee$ $\neg$ try to assassinate (x_1, y_1) $\vee$ $\neg$ ruler (y_1) $\vee$

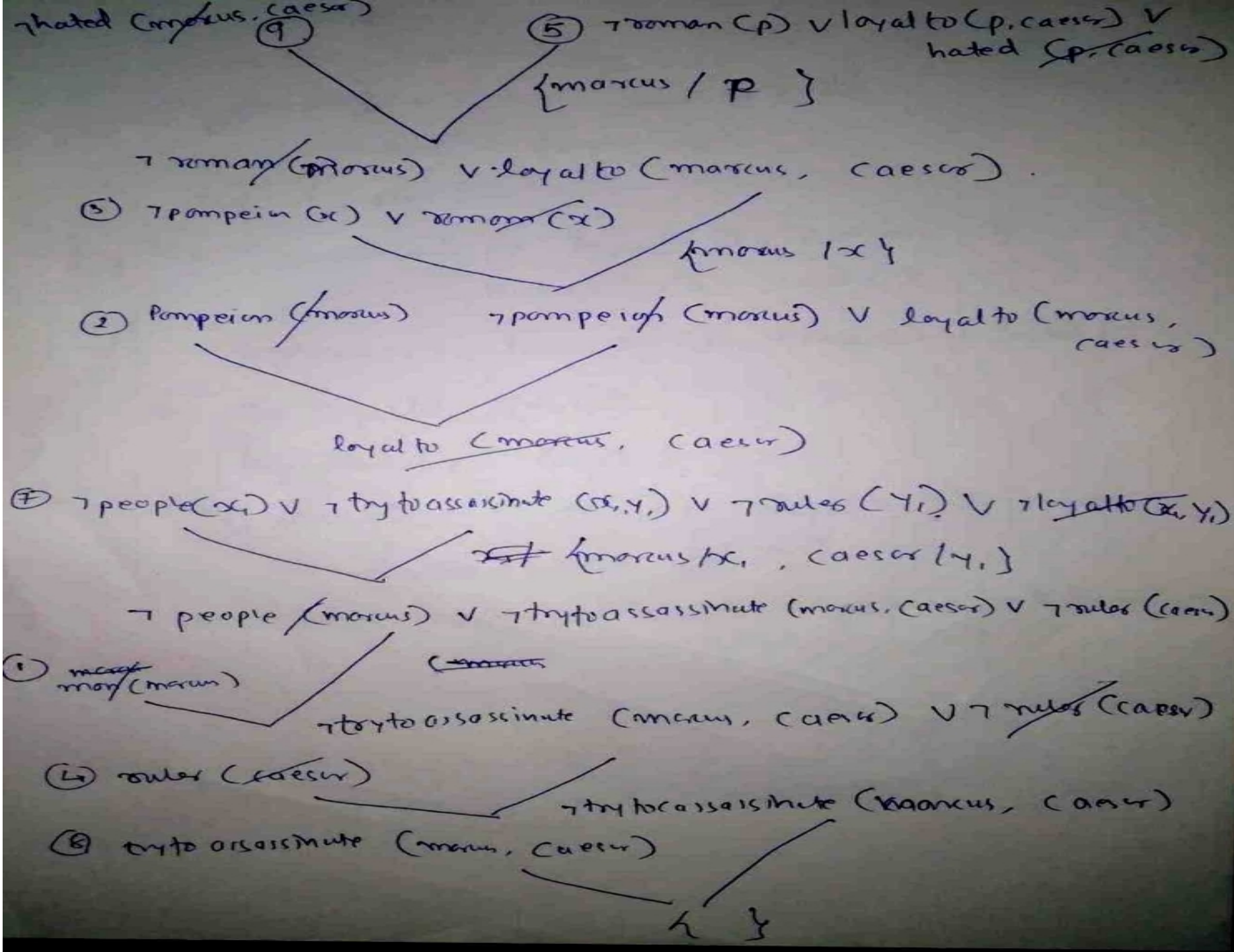
$\neg$ loyal to (x_1, y_1)

8. try to assassinate (marcus, caesar)

Prove that

hated (marcus, caesar).

$\therefore$ 9. $\neg$ hated (marcus, caesar)


¬hated (marcus, caesar)
(9)
(5) ¬roman (p) V loyal to (p, caesar) V hated (p, caesar)
{marcus / p}
¬ roman (marcus) V loyal to (marcus, caesar)
(3) ¬pompeian (x) V ¬roman (x)
{marcus / x}
(2) Pompeian (marcus)
¬pompeian (marcus) V loyal to (marcus, caesar)
loyal to (marcus, caesar)
(7) ¬people (x1) V ¬try to assassinate (x1, y1) V ¬ruler (y1) V ¬loyal to (x1, y1)
{marcus / x1, caesar / y1}
¬ people (marcus) V ¬try to assassinate (marcus, caesar) V ¬ruler (caesar)
(1)
¬try to assassinate (marcus, caesar) V ¬ruler (caesar)
(4) ruler (caesar)
¬try to assassinate (marcus, caesar)
(8) try to assassinate (marcus, caesar)
{ }

Inference Engine

- The inference engine is the component of the intelligent system in artificial intelligence, which applies logical rules to the knowledge base to infer new information from known facts.
- The first inference engine was part of the expert system.

Inference engine commonly proceeds in two modes, which are:

- **Forward chaining**
- **Backward chaining**

Horn Clause and Definite clause:

- Horn clause and definite clause are the forms of sentences, which enables knowledge base to use a more restricted and efficient inference algorithm.
- Logical inference algorithms use forward and backward chaining approaches, which require KB in the form of the **first-order definite clause.**
  - **Definite clause:** A clause which is a disjunction of literals with **exactly one positive literal** is known as a definite clause or strict horn clause.
  - **Horn clause:** A clause which is a disjunction of literals with **at most one positive literal** is known as horn clause. Hence all the definite clauses are horn clauses.
  - **Example:** $(\neg p \vee \neg q \vee k)$. It has only one positive literal k.
  - It is equivalent to $p \wedge q \rightarrow k$.

Forward Chaining

- Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine. Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.
- The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.

Forward Chaining

- It is a down-up approach, as it moves from bottom to top.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
- Forward-chaining approach is also called as data-driven as we reach to the goal using available data.
- Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

Steps for forward chaining

- Step 1:
  - We start from the already stated facts, and then, we'll subsequently choose the facts that do not have any implications at all.
- Step 2:
  - Now, we will state those facts that can be inferred from available facts with satisfied premises.
- Step 3:
  - In step 3 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.

Forward chaining Example

- If D barks and D eats bone, then D is a dog.
- If V is cold and V is sweet, then V is ice-cream.
- If D is a dog, then D is black.
- If V is ice-cream, then it is Vanilla.

- ***Derive forward chaining using the given known facts to prove Tomy is black.***
  - Tomy barks.

Example

- **"As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen."**
- Prove that "**Robert is criminal.**"

- It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)

  American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)

- Country A has some missiles. **?p Owns(A, p) ∧ Missile(p)**. It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.

  Owns(A, T1) .....(2)

  Missile(T1) .....(3)

- All of the missiles were sold to country A by Robert.

  ?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) .....(4)

- Robert is American

  American(Robert). .....(5)

- It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)

  American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)

- Country A has some missiles. **?p Owns(A, p) ∧ Missile(p)**. It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.

  Owns(A, T1) .....(2)

  Missile(T1) .....(3)

- All of the missiles were sold to country A by Robert.

  ?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) .....(4)

- Missiles are weapons.

  Missile(p) → Weapons (p) .....(5)

- Enemy of America is known as hostile.

  Enemy(p, America) →Hostile(p) .....(6)

- Country A is an enemy of America.

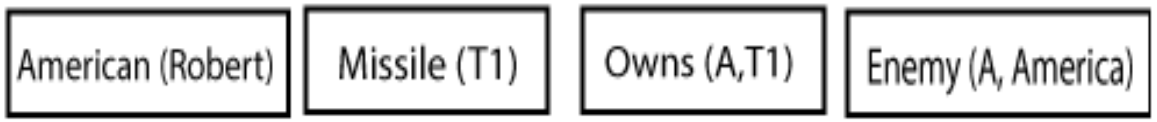
  Enemy (A, America) .....(7)

- Robert is American

  American(Robert). .....(8)

Step-1:

In the first step we will start with the known facts and will choose the sentences which do not have implications, such as: **American(Robert), Enemy(A, America), Owns(A, T1), and Missile(T1).** All these facts will be represented as below.



Step-2:

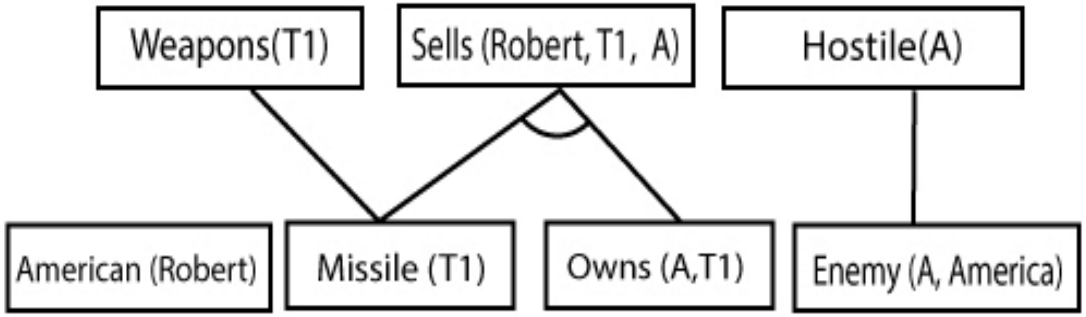
At the second step, we will see those facts which infer from available facts and with satisfied premises.

Rule-(1) does not satisfy premises, so it will not be added in the first iteration.

Rule-(2) and (3) are already added.

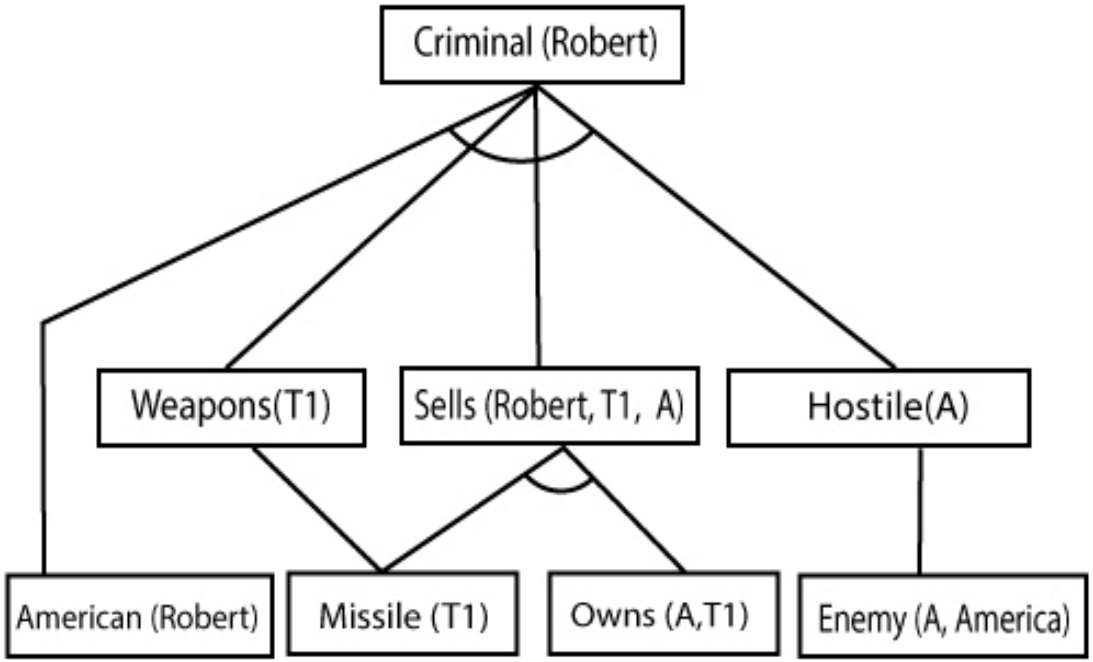
Rule-(4) satisfy with the substitution {p/T1}, **so Sells (Robert, T1, A)** is added, which infers from the conjunction of Rule (2) and (3).

Rule-(6) is satisfied with the substitution(p/A), so Hostile(A) is added and which infers from Rule-(7).



Step-3:

At step-3, as we can check Rule-(1) is satisfied with the substitution **{p/Robert, q/T1, r/A}, so we can add Criminal(Robert)** which infers all the available facts. And hence we reached our goal statement.



Backward chaining

- Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine.
- A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.

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 goals 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.

Steps for Backward chaining

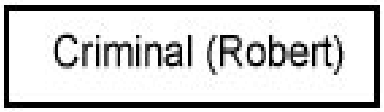
- Step 1.
  - In the first step, we'll take the Goal Fact and from the goal fact, we'll derive other facts that we'll prove true.
- Step 2:
  - We'll derive other facts from goal facts that satisfy the rules
- Step 3:
  - At step-3, we will extract further fact which infers from facts inferred in step 2.
- Step 4:
  - We'll repeat the same until we get to a certain fact that satisfies the conditions.

Example

- **American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)**
  Owns(A, T1) .....(2)
- **Missile(T1)**
- **For all p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) .....(4)**
- **Missile(p) → Weapons (p) .....(5)**
- **Enemy(p, America) →Hostile(p)**
  .....(6)
- **Enemy (A, America) .....(7)**
- **American(Robert). .....(8)**

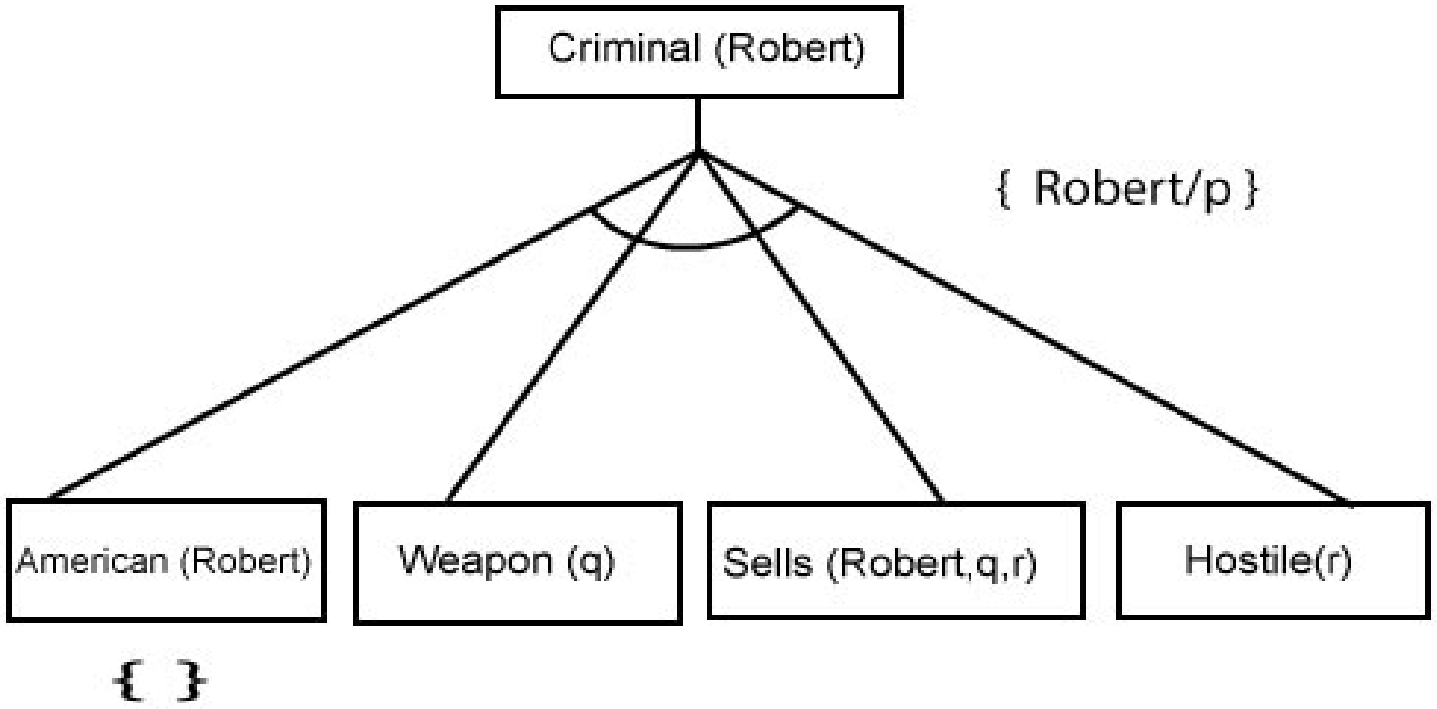
Step 1:

In the first step, we'll take the Goal Fact and from the goal fact, we'll derive other facts that we'll prove true.

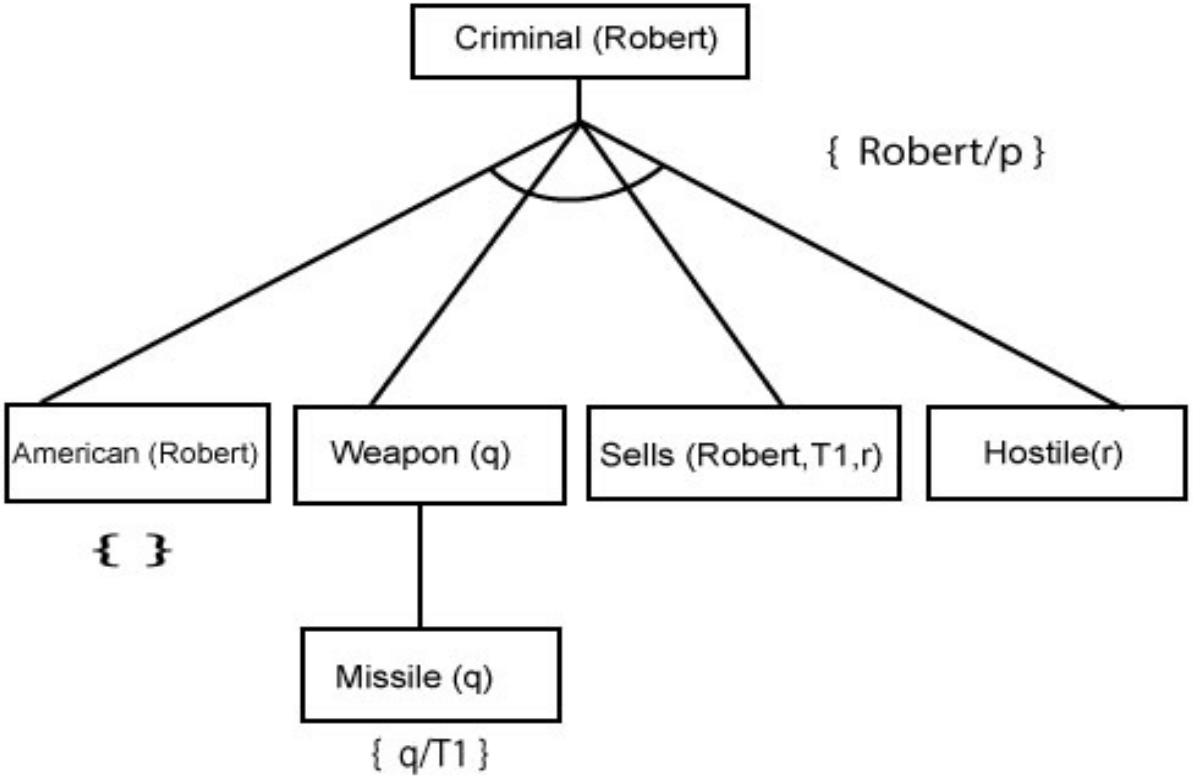


Step 2:

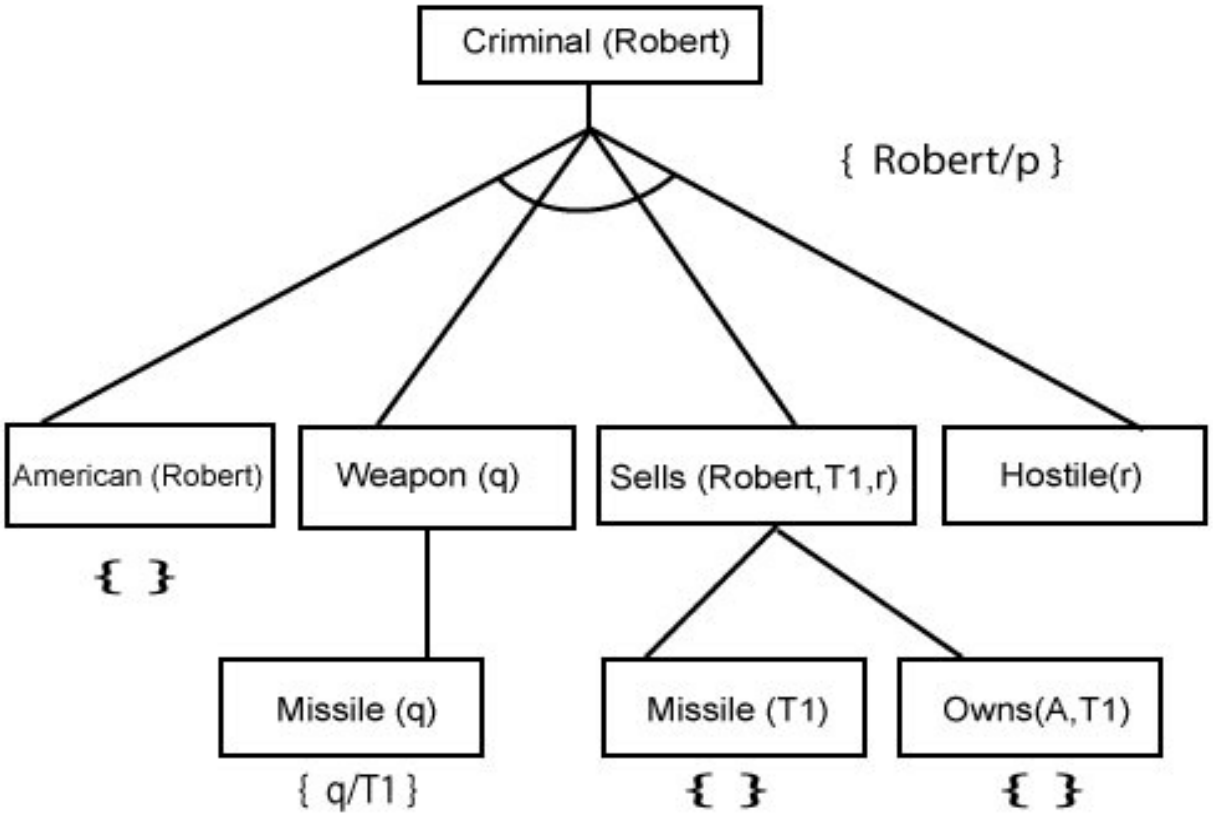
In second step, we'll derive other facts from goal facts that satisfy the rules



Step 3: At step-3, we will extract further facts which infers from facts inferred in step 2.

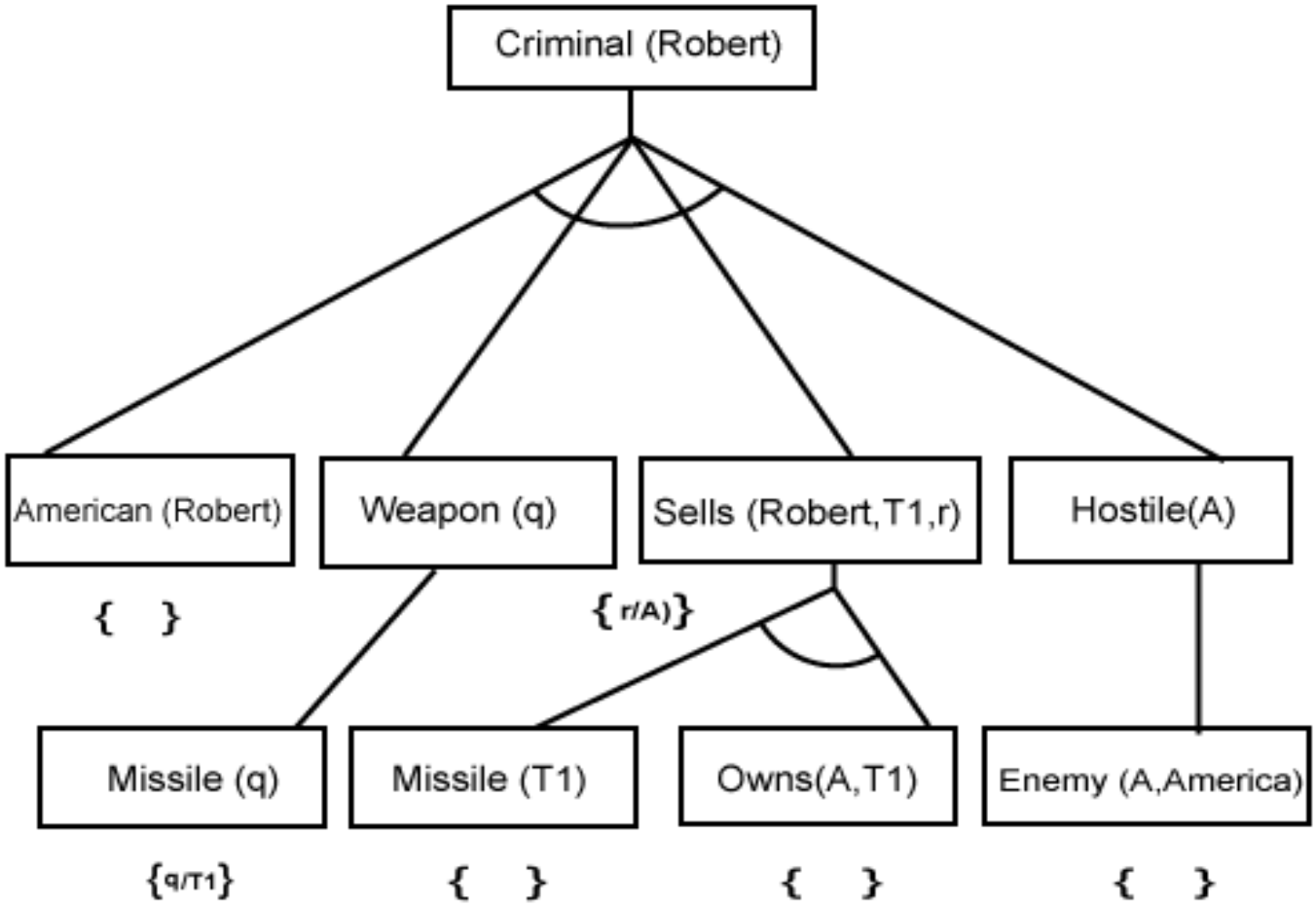


Step 4: We'll repeat the same until we get to a certain fact that satisfies the conditions.



Step 5:

Once all the facts and conditions have been derived, the iteration process stops.



Prolog

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
missile(M1).
 owns(Nono,M1).
sells(West,X,Nono) :- missile(X), owns(Nono,X).
weapon(X) :- missile(X).
 hostile(X) :- enemy(X,America).
 American(West).
 Enemy(Nono,America).
```

The following is the rule set of a simple weather forecast expert system:

1	IF	*cyclone*	THEN	*clouds*
2	IF	*anticyclone*	THEN	*clear sky*
3	IF	*pressure is low*	THEN	*cyclone*
4	IF	*pressure is high*	THEN	*anticyclone*
5	IF	*arrow is down*	THEN	*pressure is low*
6	IF	*arrow is up*	THEN	*pressure is high*

Consider the following familiar set of rules:

1	IF	*green*	THEN	*walk*
2	IF	*red*	THEN	*wait*
3	IF	*green* AND *blinking*	THEN	*hurry*
4	IF	*red* OR *green*	THEN	*traffic light works*

S. No.	Forward Chaining	Backward Chaining
1.	Forward chaining starts from known facts and applies inference rule to extract more data unit it reaches to the goal.	Backward chaining starts from the goal and works backward through inference rules to find the required facts that support the goal.
2.	It is a bottom-up approach	It is a top-down approach
3.	Forward chaining is known as data-driven inference technique as we reach to the goal using the available data.	Backward chaining is known as goal-driven technique as we start from the goal and divide into sub-goal to extract the facts.
4.	Forward chaining reasoning applies a breadth-first search strategy.	Backward chaining reasoning applies a depth-first search strategy.
5.	Forward chaining tests for all the available rules	Backward chaining only tests for few required rules.
6.	Forward chaining is suitable for the planning, monitoring, control, and interpretation application.	Backward chaining is suitable for diagnostic, prescription, and debugging application.
7.	Forward chaining can generate an infinite number of possible conclusions.	Backward chaining generates a finite number of possible conclusions.
8.	It operates in the forward direction.	It operates in the backward direction.
9.	Forward chaining is aimed for any conclusion.	Backward chaining is only aimed for the required data.