

UNIT - III

Logic and Knowledge Representation

First-Order Logic: Representation, Syntax and Semantics of First-Order Logic, Using First-Order Logic, Knowledge Engineering in First-Order Logic.

Inference in First-Order Logic: Propositional vs. First-Order Inference, Unification and Lifting, Forward Chaining, Backward Chaining, Resolution.

Knowledge Representation: Ontological Engineering, Categories and Objects, Events. Mental Events and Mental Objects, Reasoning Systems for Categories, Reasoning with Default Information.

First-Order Logic in Artificial intelligence

In the topic of Propositional logic, we have seen how to represent statements using propositional logic. But unfortunately, in propositional logic, we can only represent the facts, which are either true or false. PL is not sufficient to represent complex sentences or natural language statements. The propositional logic has very limited expressive power. Consider the following sentence, which we cannot represent using PL logic.

- "Some humans are intelligent", or
- "Sachin likes cricket."

To represent the above statements, PL logic is not sufficient, so we require some more powerful logic, such as 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:
 - a. **Objects:** A, B, people, numbers, colors, wars, theories, squares, pits, wumpus,
 - b. **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
 - c. **Function:** Father of, best friend, third inning of, end of,
- As a natural language, first-order logic also has two main parts:
 - a. **Syntax**
 - b. **Semantics**

Syntax of First-Order logic:

The syntax of FOL determines which collection of symbols is a logical expression in first-order logic. The basic syntactic elements of first-order logic are symbols. We write statements in short-hand notation in FOL.

Basic Elements of First-order logic:

Following are the basic elements of FOL syntax:

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf,
Connectives	$\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$
Equality	$=$
Quantifier	\forall, \exists

Atomic sentences:

- Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.
- We can represent atomic sentences as **Predicate (term1, term2,, term n)**.

Example: Ravi and Ajay are brothers: \Rightarrow Brothers(Ravi, Ajay).

Chinky is a cat: \Rightarrow cat (Chinky).

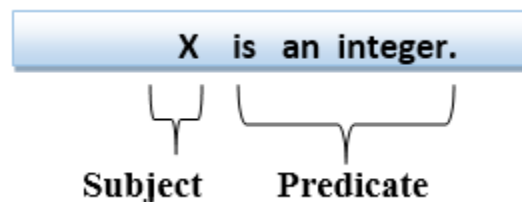
Complex Sentences:

- Complex sentences are made by combining atomic sentences using connectives.

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.

Consider the statement: "x is an integer.", it consists of two parts, the first part x is the subject of the statement and second part "is an integer," is known as a predicate.



Quantifiers in First-order logic:

- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
- These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifier:
 - a. **Universal Quantifier, (for all, everyone, everything)**
 - b. **Existential quantifier, (for some, at least one).**

Universal Quantifier:

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

The Universal quantifier is represented by a symbol \forall , which resembles an inverted A.

Note: In universal quantifiers we use the implication " \rightarrow ".

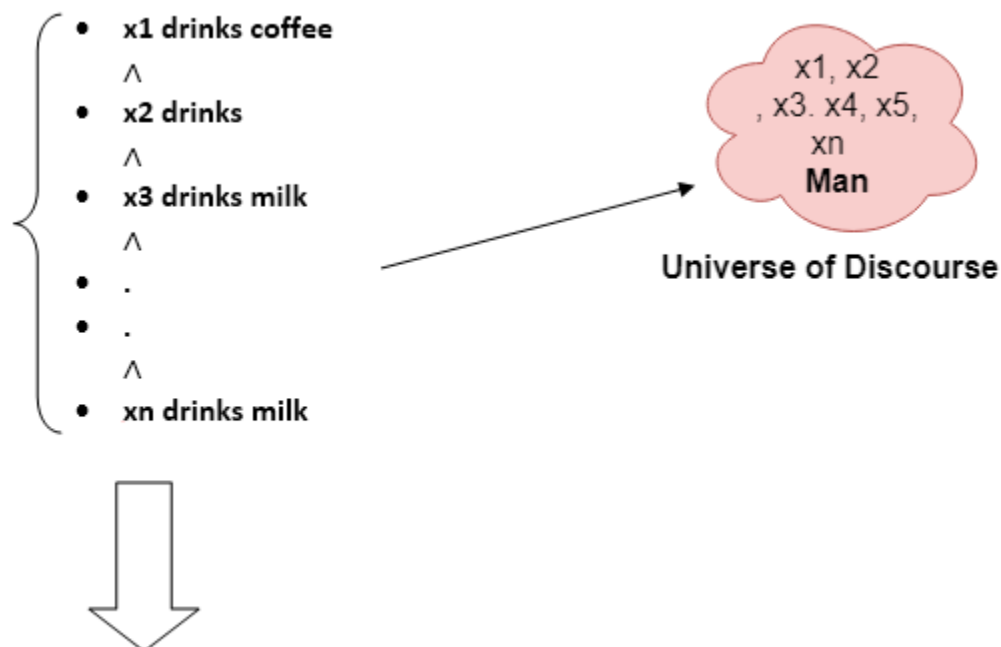
If x is a variable, then $\forall x$ is read as:

- **For all x**
- **For each x**
- **For every x.**

Example:

All men drink coffee.

Let a variable x which refers to a cat so all x can be represented in UOD as below:



So in shorthand notation, we can write it as :

$\forall x \text{ man}(x) \rightarrow \text{drink}(x, \text{coffee}).$

It will be read as: There are all x where x is a man who drinks coffee.

Existential Quantifier:

Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something.

It is denoted by the logical operator \exists , which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

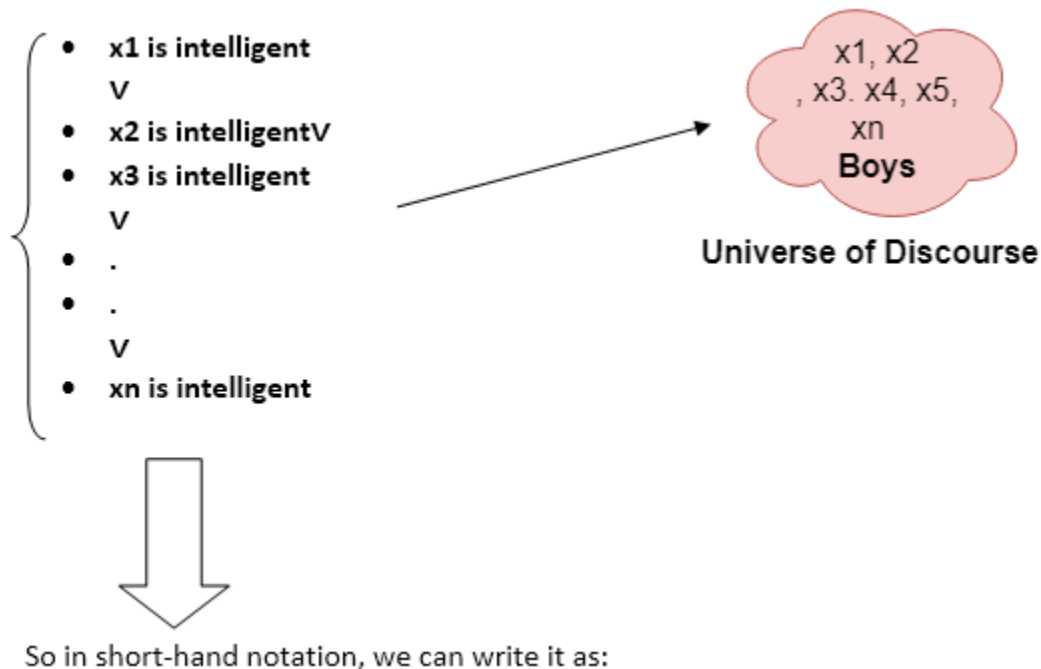
Note: In Existential quantifier we always use AND or Conjunction symbol (\wedge).

If x is a variable, then existential quantifier will be $\exists x$ or $\exists(x)$. And it will be read as:

- **There exists a 'x.'**
- **For some 'x.'**
- **For at least one 'x.'**

Example:

Some boys are intelligent.



$\exists x: \text{boys}(x) \wedge \text{intelligent}(x)$

It will be read as: There are some x where x is a boy who is intelligent.

Points to remember:

- The main connective for universal quantifier \forall is implication \rightarrow .
- The main connective for existential quantifier \exists is and \wedge .

Properties of Quantifiers:

- In universal quantifier, $\forall x \forall y$ is similar to $\forall y \forall x$.
- In Existential quantifier, $\exists x \exists y$ is similar to $\exists y \exists x$.
- $\exists x \forall y$ is not similar to $\forall y \exists x$.

Some Examples of FOL using quantifier:

1. All birds fly.

In this question the predicate is "**fly(bird).**"

And since there are all birds who fly, it will be represented as follows.

$$\forall x \text{ bird}(x) \rightarrow \text{fly}(x).$$

2. Every man respects his parents.

In this question, the predicate is "**respect(x, y),**" where **x=man, and y= parent.**

Since there is every man so will use \forall , and it will be represented as follows:

$$\forall x \text{ man}(x) \rightarrow \text{respects}(x, \text{parent}).$$

3. Some boys play cricket.

In this question, the predicate is "**play(x, y),**" where **x= boys, and y= game.** Since there are some boys so we will use \exists , **and it will be represented as:**

$$\exists x \text{ boys}(x) \rightarrow \text{play}(x, \text{cricket}).$$

4. Not all students like both Mathematics and Science.

In this question, the predicate is "**like(x, y),**" where **x= student, and y= subject.**

Since there are not all students, so we will use \forall **with negation, so** following representation for this:

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

5. Only one student failed in Mathematics.

In this question, the predicate is "**failed(x, y),**" where **x= student, and y= subject.**

Since there is only one student who failed in Mathematics, so we will use following representation for this:

$$\exists (x) [\text{student}(x) \rightarrow \text{failed}(x, \text{Mathematics}) \wedge \forall (y) [\neg(x=y) \wedge \text{student}(y) \rightarrow \neg \text{failed}(y, \text{Mathematics})].$$

Free and Bound Variables:

The quantifiers interact with variables which appear in a suitable way. There are two types of variables in First-order logic which are given below:

Free Variable: A variable is said to be a free variable in a formula if it occurs outside the scope of the quantifier.

Example: $\forall x \exists (y)[P(x, y, z)]$, where z is a free variable.

Bound Variable: A variable is said to be a bound variable in a formula if it occurs within the scope of the quantifier.

Example: $\forall x [A(x) B(y)]$, here x and y are the bound variables.

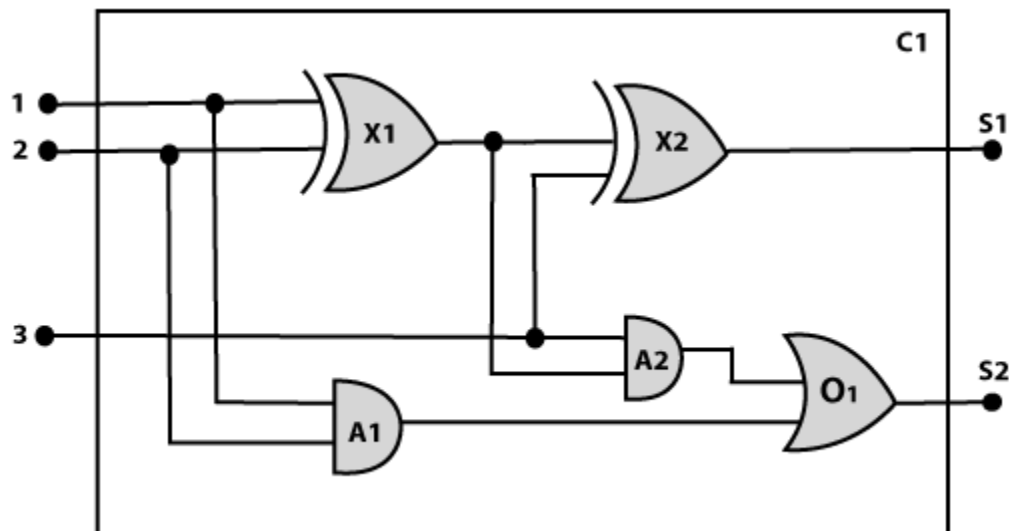
Knowledge Engineering in First-order logic
What is knowledge-engineering?

The process of constructing a knowledge-base in first-order logic is called knowledge- engineering. In **knowledge-engineering**, someone who investigates a particular domain, learns an important concept of that domain, and generates a formal representation of the objects, is known as a knowledge **engineer**.

In this topic, we will understand the Knowledge engineering process in an electronic circuit domain, which is already familiar. This approach is mainly suitable for creating a special-purpose **knowledge base**.

The knowledge-engineering process:

Following are some main steps of the knowledge-engineering process. Using these steps, we will develop a knowledge base which will allow us to reason about digital circuit (**One-bit full adder**) which is given below



1. Identify the task:

The first step of the process is to identify the task, and for the digital circuit, there are various reasoning tasks.

At the first level or highest level, we will examine the functionality of the circuit:

- Does the circuit add properly?
- What will be the output of gate A2, if all the inputs are high?

At the second level, we will examine the circuit structure details such as:

- Which gate is connected to the first input terminal?
- Does the circuit have feedback loops?

2. Assemble the relevant knowledge:

In the second step, we will assemble the relevant knowledge which is required for digital circuits. So for digital circuits, we have the following required knowledge:

- Logic circuits are made up of wires and gates.
- Signal flows through wires to the input terminal of the gate, and each gate produces the corresponding output which flows further.
- In this logic circuit, there are four types of gates used: **AND, OR, XOR, and NOT**.
- All these gates have one output terminal and two input terminals (except NOT gate, it has one input terminal).

3. Decide on vocabulary:

The next step of the process is to select functions, predicates, and constants to represent the circuits, terminals, signals, and gates. Firstly we will distinguish the gates from each other and from other objects. Each gate is represented as an object which is named by a constant, such as **Gate(X1)**. The functionality of each gate is determined by its type, which is taken as constants such as **AND, OR, XOR, or NOT**. Circuits will be identified by a predicate: **Circuit (C1)**.

For the terminal, we will use predicate: **Terminal(x)**.

For gate input, we will use the function **In(1, X1)** for denoting the first input terminal of the gate, and for output terminal we will use **Out (1, X1)**.

The function **Arity(c, i, j)** is used to denote that circuit c has i input, j output.

The connectivity between gates can be represented by the predicate **Connect(Out(1, X1), In(1, X1))**.

We use a unary predicate **On (t)**, which is true if the signal at a terminal is on.

4. Encode general knowledge about the domain:

To encode the general knowledge about the logic circuit, we need some following rules:

- If two terminals are connected then they have the same input signal, it can be represented as:
 1. $\forall t1, t2 \text{ Terminal}(t1) \wedge \text{Terminal}(t2) \wedge \text{Connect}(t1, t2) \rightarrow \text{Signal}(t1) = \text{Signal}(2)$.
- Signal at every terminal will have either value 0 or 1, it will be represented as:
 1. $\forall t \text{ Terminal}(t) \rightarrow \text{Signal}(t) = 1 \vee \text{Signal}(t) = 0$.
- Connect predicates are commutative:
 1. $\forall t1, t2 \text{ Connect}(t1, t2) \rightarrow \text{Connect}(t2, t1)$.
- Representation of types of gates:
 1. $\forall g \text{ Gate}(g) \wedge r = \text{Type}(g) \rightarrow r = \text{OR} \vee r = \text{AND} \vee r = \text{XOR} \vee r = \text{NOT}$.
- Output of AND gate will be zero if and only if any of its input is zero.

1. $\forall g \text{ Gate}(g) \wedge \text{Type}(g) = \text{AND} \rightarrow \text{Signal}(\text{Out}(1, g)) = 0 \Leftrightarrow \exists n \text{ Signal}(\text{In}(n, g)) = 0$.
- Output of OR gate is 1 if and only if any of its input is 1:
1. $\forall g \text{ Gate}(g) \wedge \text{Type}(g) = \text{OR} \rightarrow \text{Signal}(\text{Out}(1, g)) = 1 \Leftrightarrow \exists n \text{ Signal}(\text{In}(n, g)) = 1$
- Output of XOR gate is 1 if and only if its inputs are different:
1. $\forall g \text{ Gate}(g) \wedge \text{Type}(g) = \text{XOR} \rightarrow \text{Signal}(\text{Out}(1, g)) = 1 \Leftrightarrow \text{Signal}(\text{In}(1, g)) \neq \text{Signal}(\text{In}(2, g))$.
- Output of NOT gate is invert of its input:
1. $\forall g \text{ Gate}(g) \wedge \text{Type}(g) = \text{NOT} \rightarrow \text{Signal}(\text{In}(1, g)) \neq \text{Signal}(\text{Out}(1, g))$.
- All the gates in the above circuit have two inputs and one output (except NOT gate).
1. $\forall g \text{ Gate}(g) \wedge \text{Type}(g) = \text{NOT} \rightarrow \text{Arity}(g, 1, 1)$
2. $\forall g \text{ Gate}(g) \wedge r = \text{Type}(g) \wedge (r = \text{AND} \vee r = \text{OR} \vee r = \text{XOR}) \rightarrow \text{Arity}(g, 2, 1)$.
- All gates are logic circuits:
1. $\forall g \text{ Gate}(g) \rightarrow \text{Circuit}(g)$.

5. Encode a description of the problem instance:

Now we encode the problem of circuit C1, firstly we categorize the circuit and its gate components. This step is easy if the ontology about the problem is already thought through. This step involves the writing of simple atomic sentences of instances of concepts, which is known as ontology.

For the given circuit C1, we can encode the problem instance in atomic sentences as below:

Since in the circuit there are two XOR, two AND, and one OR gate so atomic sentences for these gates will be:

1. For XOR gate: $\text{Type}(x1) = \text{XOR}, \text{Type}(x2) = \text{XOR}$
2. For AND gate: $\text{Type}(A1) = \text{AND}, \text{Type}(A2) = \text{AND}$
3. For OR gate: $\text{Type}(O1) = \text{OR}$.

And then represent the connections between all the gates.

Note: Ontology defines a particular theory of the nature of existence.

6. Pose queries to the inference procedure and get answers:

In this step, we will find all the possible sets of values of all the terminals for the adder circuit. The first query will be:

What should be the combination of input which would generate the first output of circuit C1, as 0 and a second output to be 1?

1. $\exists i1, i2, i3 \text{ Signal}(\text{In}(1, C1)) = i1 \wedge \text{Signal}(\text{In}(2, C1)) = i2 \wedge \text{Signal}(\text{In}(3, C1)) = i3$
2. $\wedge \text{Signal}(\text{Out}(1, C1)) = 0 \wedge \text{Signal}(\text{Out}(2, C1)) = 1$

7. Debug the knowledge base:

Now we will debug the knowledge base, and this is the last step of the complete process. In this step, we will try to debug the issues of knowledge base.

In the knowledge base, we may have omitted assertions like $1 \neq 0$.

Inference in First-Order Logic

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 $F[a/x]$, it means to substitute a constant "a" in place of variable "x".

Note: First-order logic is capable of expressing facts about some or all objects in the universe.

Equality:

First-Order logic does not only use predicates 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: $\neg(x=y)$ which is equivalent to $x \neq y$.

FOL inference rules for quantifier:

As propositional logic we also have inference rules in first-order logic, so following are some basic inference rules in FOL:

- **Universal Generalization**
- **Universal Instantiation**
- **Existential Instantiation**
- **Existential introduction**

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 $\forall x P(x)$.

$$\frac{P(c)}{\forall x P(x)}$$

- It can be represented as: $\forall x P(x)$.
- 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.**

$$\frac{\forall x P(x)}{P(c)}$$

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

Example:1.

IF "Every person like ice-cream" $\Rightarrow \forall x P(x)$ so we can infer that

"John likes ice-cream" $\Rightarrow P(c)$

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:

$\forall x \text{king}(x) \wedge \text{greedy}(x) \rightarrow \text{Evil}(x),$

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 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 the old KB, but it will be satisfiable if the 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.

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

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

Example:

From the given sentence: $\exists x \text{Crown}(x) \wedge \text{OnHead}(x, \text{John}),$

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**.
- Existential instantiation is a special case of the Skolemization **process**.

4. Existential 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.

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

- It can be represented as: $\exists x P(x)$
- **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 a 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 $\text{SUBST}(\theta, \text{pi}') = \text{SUBST}(\theta, \text{pi})$, it can be represented as:

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

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.

1. Here let say, p1' is king(John) p1 is king(x)
2. p2' is Greedy(y) p2 is Greedy(x)
3. θ is {x/John, y/John} q is evil(x)
4. $\text{SUBST}(\theta, q)$.

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, $\Psi_1\sigma = \Psi_2\sigma$, then it can be expressed as **UNIFY**(Ψ_1, Ψ_2).
- **Example: Find the MGU for Unify{King(x), King(John)}**

Let $\Psi_1 = \text{King}(x)$, $\Psi_2 = \text{King}(\text{John})$,

Substitution $\theta = \{\text{John}/x\}$ is a unifier for these atoms and applying this substitution, 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 fails if the expressions do not match with each other.
- The substitution variables are called Most General Unifier or MGU.

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 symbols must be the same, atoms or expressions with different predicate symbols 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:

Algorithm: Unify(Ψ_1, Ψ_2)

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 Ψ_1 is a variable,
 - a. then if Ψ_1 occurs in Ψ_2 , then return FAILURE
 - b. Else return $\{(\Psi_2/\Psi_1)\}$.
- c) Else if Ψ_2 is a variable,
 - a. If Ψ_2 occurs in Ψ_1 then return FAILURE,
 - b. Else return $\{(\Psi_1/\Psi_2)\}$.
- 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 i th element of Ψ_1 and i th element of Ψ_2 , and put the result into S.
- b) If S = failure then returns Failure
- c) If $S \neq \text{NIL}$ then do,
 - a. Apply S to the remainder of both L1 and L2.
 - b. $\text{SUBST} = \text{APPEND}(S, \text{SUBST})$.

Step.6: Return SUBST.

Implementation of the Algorithm

Step.1: Initialize the substitution set to be empty.

Step.2: Recursively unify atomic sentences:

- a. Check for Identical expression matches.
- b. If one expression is a variable v_i , and the other is a term to which does not contain variable v_i , then:
 - a. Substitute t_i / v_i in the existing substitutions
 - b. Add t_i / v_i to the substitution setlist.
 - c. If both the expressions are functions, then the function name must be similar, and the number of arguments must be the same in both the expressions.

For each pair of the following atomic sentences find the most general unifier (If exist).

1. Find the MGU of $\{p(f(a), g(Y)) \text{ and } p(X, X)\}$

Sol: $S_0 \Rightarrow$ Here, $\Psi_1 = p(f(a), g(Y))$, and $\Psi_2 = p(X, X)$

$\text{SUBST } \theta = \{f(a) / X\}$

$S_1 \Rightarrow \Psi_1 = p(f(a), g(Y))$, and $\Psi_2 = p(f(a), f(a))$

$\text{SUBST } \theta = \{f(a) / g(y)\}$, **Unification failed.**

Unification is not possible for these expressions.

2. Find the MGU of $\{p(b, X, f(g(Z))) \text{ and } p(Z, f(Y), f(Y))\}$

Here, $\Psi_1 = p(b, X, f(g(Z)))$, and $\Psi_2 = p(Z, f(Y), f(Y))$

$S_0 \Rightarrow \{p(b, X, f(g(Z))); p(Z, f(Y), f(Y))\}$

$\text{SUBST } \theta = \{b/Z\}$

$S_1 \Rightarrow \{p(b, X, f(g(b))); p(b, f(Y), f(Y))\}$

$\text{SUBST } \theta = \{f(Y) / X\}$

$S_2 \Rightarrow \{p(b, f(Y), f(g(b))); p(b, f(Y), f(Y))\}$

$\text{SUBST } \theta = \{g(b) / Y\}$

$S_2 \Rightarrow \{p(b, f(g(b)), f(g(b))); p(b, f(g(b)), f(g(b)))\}$ **Unified Successfully.**

And Unifier = { b/Z , $f(Y)/X$, $g(b)/Y$ }.

3. Find the MGU of { $p(X, X)$, and $p(Z, f(Z))$ }

Here, $\Psi_1 = \{p(X, X)\}$, and $\Psi_2 = \{p(Z, f(Z))\}$

$S_0 \Rightarrow \{p(X, X), p(Z, f(Z))\}$

SUBST $\theta = \{X/Z\}$

$S_1 \Rightarrow \{p(Z, Z), p(Z, f(Z))\}$

SUBST $\theta = \{f(Z)/Z\}$, **Unification Failed.**

Hence, unification is not possible for these expressions.

4. Find the MGU of UNIFY($\text{prime}(11)$, $\text{prime}(y)$)

Here, $\Psi_1 = \{\text{prime}(11)\}$, and $\Psi_2 = \{\text{prime}(y)\}$

$S_0 \Rightarrow \{\text{prime}(11), \text{prime}(y)\}$

SUBST $\theta = \{11/y\}$

$S_1 \Rightarrow \{\text{prime}(11), \text{prime}(11)\}$, **Successfully unified.**

Unifier: $\{11/y\}$.

5. Find the MGU of $Q(a, g(x, a), f(y))$, $Q(a, g(f(b), a), x)$

Here, $\Psi_1 = \{Q(a, g(x, a), f(y))\}$, and $\Psi_2 = \{Q(a, g(f(b), a), x)\}$

$S_0 \Rightarrow \{Q(a, g(x, a), f(y)); Q(a, g(f(b), a), x)\}$

SUBST $\theta = \{f(b)/x\}$

$S_1 \Rightarrow \{Q(a, g(f(b), a), f(y)); Q(a, g(f(b), a), f(b))\}$

SUBST $\theta = \{b/y\}$

$S_1 \Rightarrow \{Q(a, g(f(b), a), f(b)); Q(a, g(f(b), a), f(b))\}$, **Successfully Unified.**

Unifier: $[a/a, f(b)/x, b/y]$.

6. UNIFY($\text{knows}(\text{Richard}, x)$, $\text{knows}(\text{Richard}, \text{John})$)

Here, $\Psi_1 = \{\text{knows}(\text{Richard}, x)\}$, and $\Psi_2 = \{\text{knows}(\text{Richard}, \text{John})\}$

$S_0 \Rightarrow \{\text{knows}(\text{Richard}, x); \text{knows}(\text{Richard}, \text{John})\}$

SUBST $\theta = \{\text{John}/x\}$

$S_1 \Rightarrow \{\text{knows}(\text{Richard}, \text{John}); \text{knows}(\text{Richard}, \text{John})\}$, **Successfully Unified.**

Unifier: $\{\text{John}/x\}$.

Forward Chaining and backward chaining in AI

In artificial intelligence, forward and backward chaining is one of the important topics, but before understanding forward and backward chaining let's first understand where these two terms came from.

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:

- a. **Forward chaining**
- b. **Backward chaining**

Horn Clause and Definite clause:

Horn clause and definite clause are the forms of sentences, which enables the 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 $p \wedge q \rightarrow k$.

A. 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 starts 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 adds their conclusion to the known facts. This process repeats until the problem is solved.

Properties of 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 reaching the goal state.
- Forward-chaining approach is also called data-driven as we reach the goal using available data.
- Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

Consider the following famous example which we will use in both approaches:

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 a criminal."

To solve the above problem, first, we will convert all the above facts into first-order definite clauses, and then we will use a forward-chaining algorithm to reach the goal.

Facts Conversion into FOL:

- It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)
American (p) \wedge weapon(q) \wedge sells (p, q, r) \wedge hostile(r) \rightarrow Criminal(p) ... (1)
- Country A has some missiles. **?p Owns(A, p) \wedge 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) \wedge Owns (A, p) \rightarrow Sells (Robert, p, A) (4)
- Missiles are weapons.
Missile(p) \rightarrow Weapons (p) (5)
- The Enemy of America is known as hostile.
Enemy(p, America) \rightarrow Hostile(p) (6)
- Country A is an enemy of America.
Enemy (A, America) (7)
- Robert is American
American(Robert). (8)

Forward chaining proof:

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:

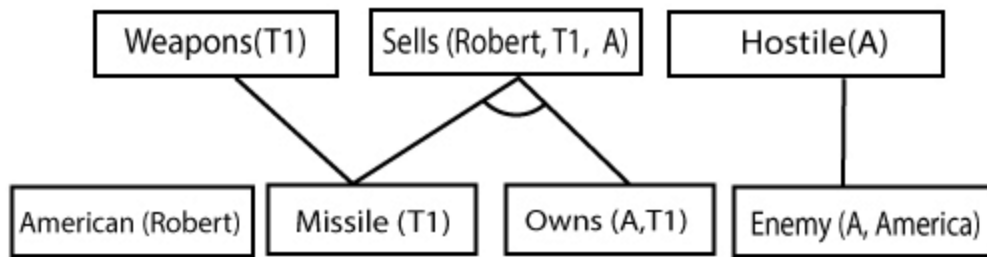
At the second step, we will see those facts which infer from available facts and with satisfied premises.

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

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

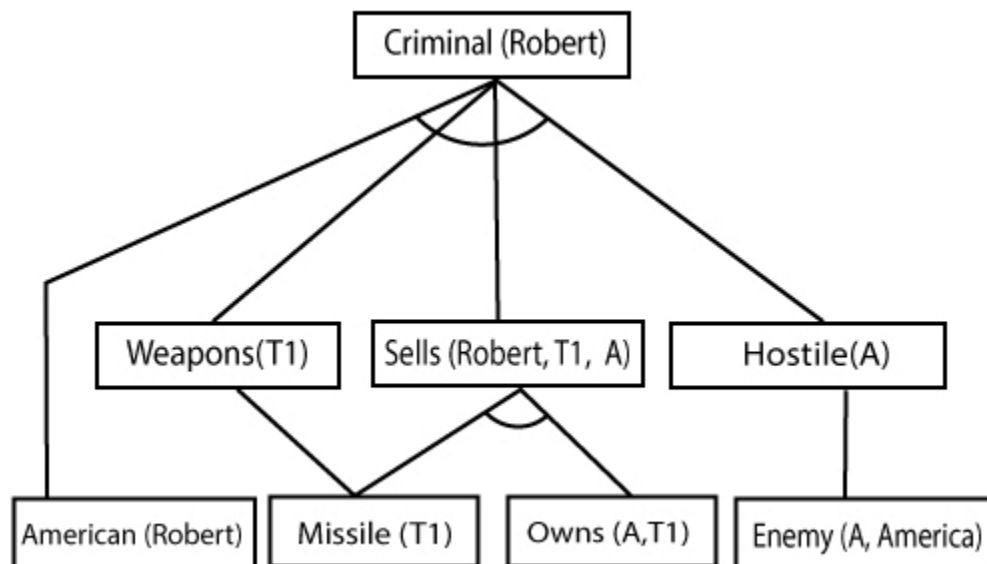
Rule-(4) satisfies with the substitution {p/T1}, so **Sells (Robert, T1, A)** is added, which infers from the conjunction of Rule (2) and (3).

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 $\{p/\text{Robert}, q/T1, r/A\}$, so we can add **Criminal(Robert)** which infers all the available facts. And hence we reached our goal statement.



Hence it is proved that Robert is Criminal using a forward chaining approach.

B. 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.

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 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.

Example:

In backward-chaining, we will use the same above example, and will rewrite all the rules.

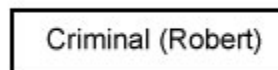
- $\text{American}(p) \wedge \text{weapon}(q) \wedge \text{sells}(p, q, r) \wedge \text{hostile}(r) \rightarrow \text{Criminal}(p) \dots(1)$
- $\text{Owns}(A, T1) \dots\dots\dots(2)$
- $\text{Missile}(T1)$
- $?p \text{ Missiles}(p) \wedge \text{Owns}(A, p) \rightarrow \text{Sells}(\text{Robert}, p, A) \dots\dots(4)$
- $\text{Missile}(p) \rightarrow \text{Weapons}(p) \dots\dots\dots(5)$
- $\text{Enemy}(p, \text{America}) \rightarrow \text{Hostile}(p) \dots\dots\dots(6)$
- $\text{Enemy}(A, \text{America}) \dots\dots\dots(7)$
- $\text{American}(\text{Robert}). \dots\dots\dots(8)$

Backward-Chaining proof:

In Backward chaining, we will start with our goal predicate, which is **Criminal(Robert)**, and then infer further rules.

Step-1:

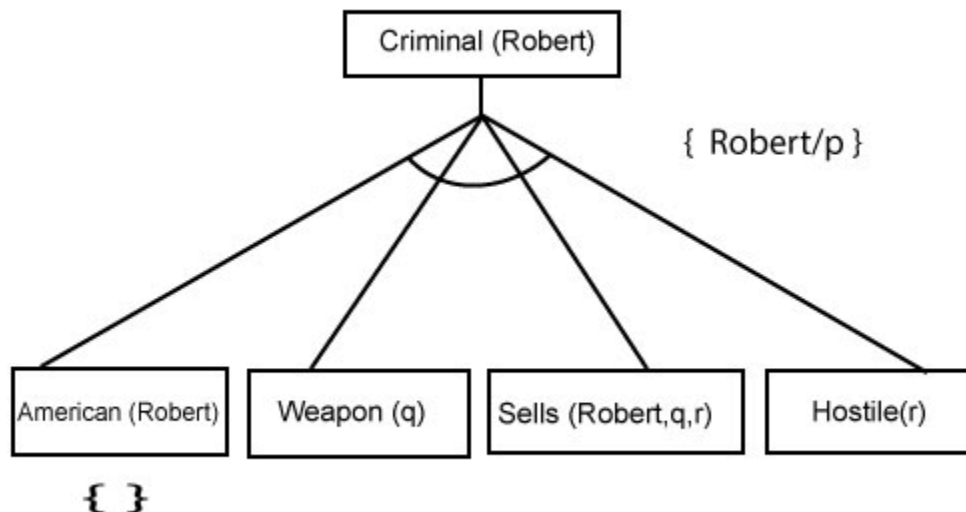
At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is "Robert is Criminal," so the following is the predicate of it.



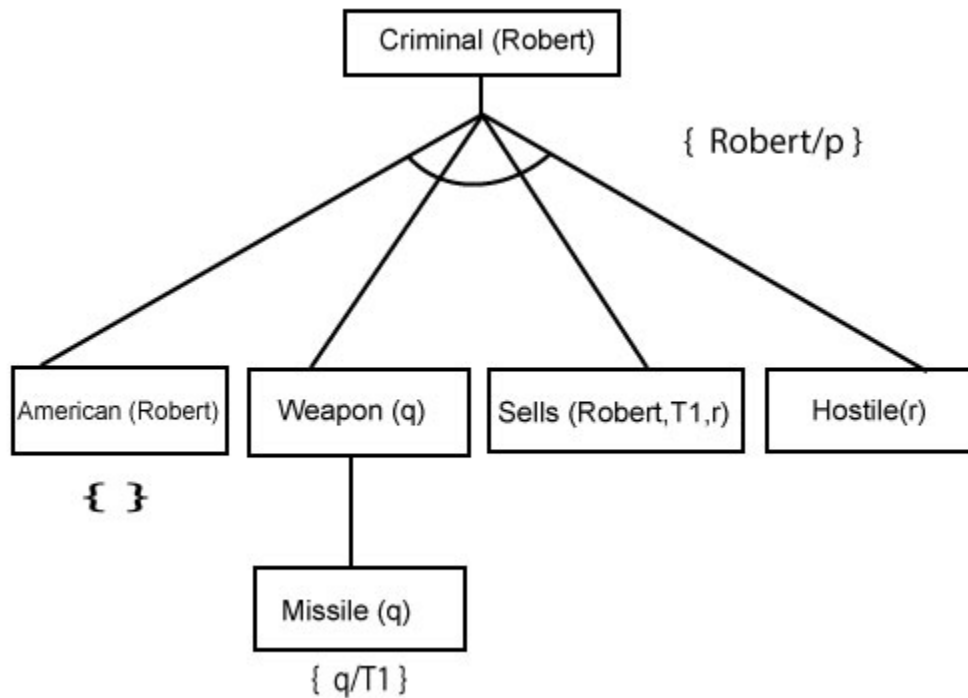
Step-2:

At the second step, we will infer other facts from the goal fact which satisfies the rules. So as we can see in Rule-1, the goal predicate Criminal (Robert) is present with substitution $\{ \text{Robert}/p \}$. So we will add all the conjunctive facts below the first level and will replace p with Robert.

Here we can see that American (Robert) is a fact, so it is proved here.

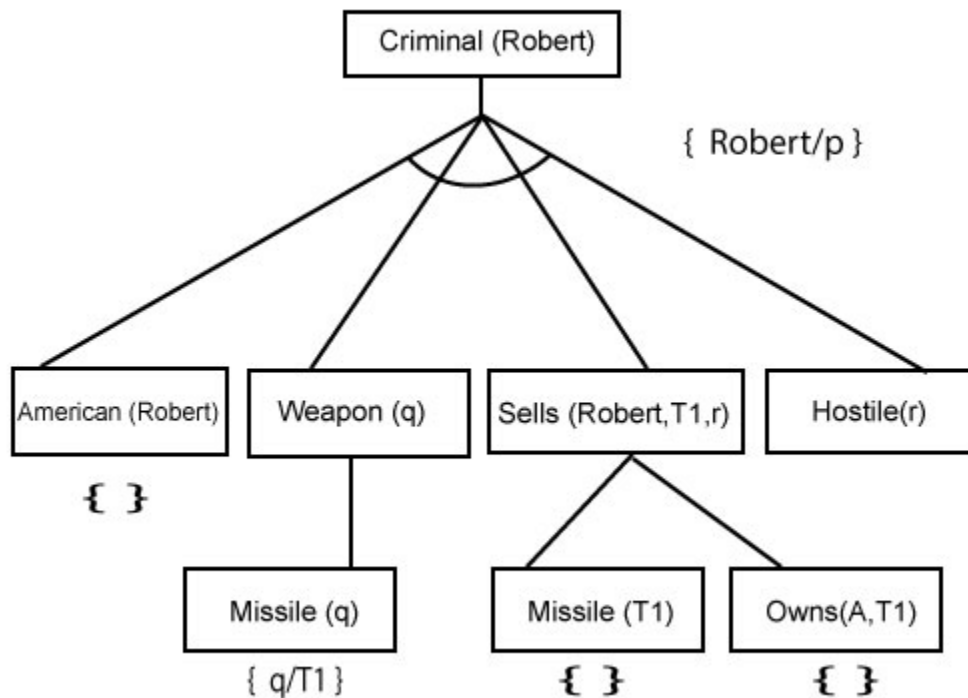


Step-3: At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q.



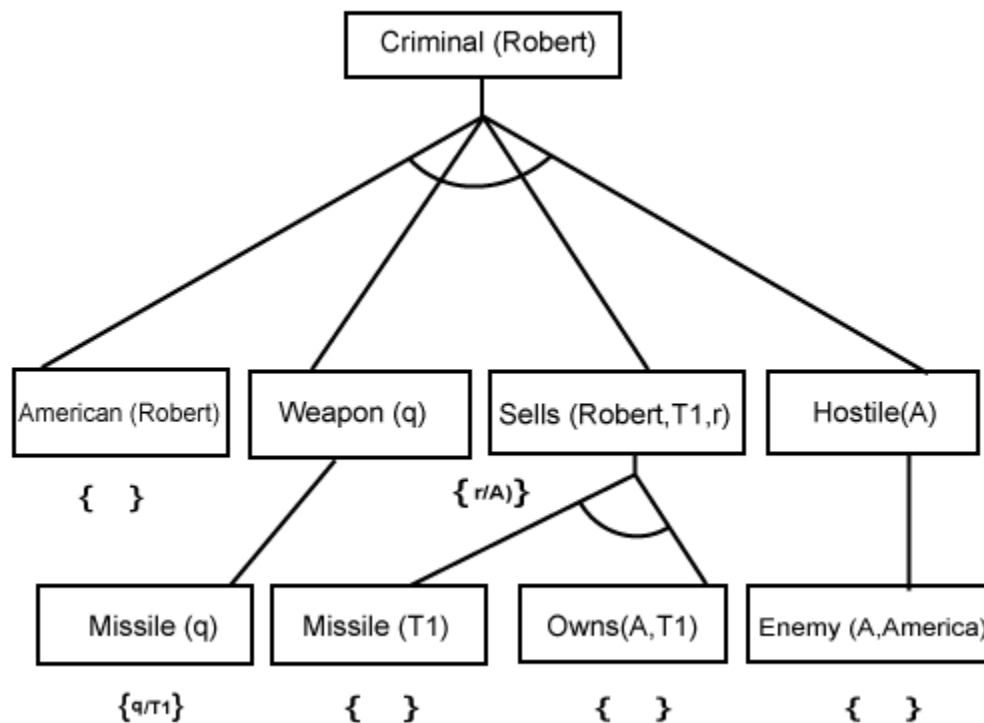
Step-4:

At step-4, we can infer facts Missile(T1) and Owns(A, T1) from Sells(Robert, T1, r) which satisfies the **Rule- 4**, with the substitution of A in place of r. So these two statements are proved here.



Step-5:

At step-5, we can infer the fact **Enemy(A, America)** from **Hostile(A)** which satisfies Rule- 6. And hence all the statements are proved true using backward chaining.



Difference between backward chaining and forward chaining

Following is the difference between the forward chaining and backward chaining:

- Forward chaining as the name suggests, starts from the known facts and moves forward by applying inference rules to extract more data, and it continues until it reaches the goal, whereas backward chaining starts from the goal, moves backward by using inference rules to determine the facts that satisfy the goal.
- Forward chaining is called a **data-driven** inference technique, whereas backward chaining is called a **goal-driven** inference technique.
- Forward chaining is known as the **down-up** approach, whereas backward chaining is known as a **top-down** approach.
- Forward chaining uses **breadth-first search** strategy, whereas backward chaining uses **depth-first search** strategy.
- Forward and backward chaining both apply **Modus ponens** inference rule.
- Forward chaining can be used for tasks such as **planning, design process monitoring, diagnosis, and classification**, whereas backward chaining can be used for **classification and diagnosis tasks**.
- Forward chaining can be like an exhaustive search, whereas backward chaining tries to avoid the unnecessary path of reasoning.
- In forward-chaining there can be various ASK questions from the knowledge base, whereas in backward chaining there can be fewer ASK questions.
- Forward chaining is slow as it checks for all the rules, whereas backward chaining is fast as it checks few required rules only.

S. No.	Forward Chaining	Backward Chaining
1.	Forward chaining starts from known facts and applies inference rules to extract more data until it reaches 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 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 applications.
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.

Resolution in FOL

Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e., proofs by contradictions. It was invented by Mathematician John Alan Robinson in 1965.

Resolution is used, if 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**.

Clause: Disjunction of literals (an atomic sentence) is called a **clause**. It is also known as a unit clause.

Conjunctive Normal Form: A sentence represented as a conjunction of clauses is said to be **conjunctive normal form** or **CNF**.

Note: To better understand this topic, firstly learn the FOL in AI.

The resolution inference rule:

The resolution rule for first-order logic is simply a lifted version of the propositional rule. Resolution can resolve two clauses if they contain complementary literals, which are assumed to be standardized apart so that they share no variables.

$$\frac{l_1 \vee \dots \vee l_k \quad m_1 \vee \dots \vee m_n}{\text{SUBST}(\theta, l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n)}$$

Where l_i and m_j are complementary literals.

This rule is also called the **binary resolution rule** because it only resolves exactly two literals.

Example:

We can resolve two clauses which are given below:

[Animal (g(x) \vee Loves (f(x), x)] and [\neg Loves(a, b) \vee \neg Kills(a, b)]

Where two complementary literals are: **Loves (f(x), x) and \neg Loves (a, b)**

These literals can be unified with unifier $\theta = [a/f(x), \text{ and } b/x]$, and it will generate a resolvent clause:

[Animal (g(x) \vee \neg Kills(f(x), x)].

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).

To better understand all the above steps, we will take an example in which we will apply resolution.

Example:

- a. **John likes all kinds of food.**
- b. **Apple and vegetable are food**
- c. **Anything anyone eats and is not killed is food.**
- d. **Anil eats peanuts and still alive**
- e. **Harry eats everything that Anil eats.**

Prove by resolution that:

f. **John likes peanuts.**

Step-1: Conversion of Facts into FOL

In the first step we will convert all the given statements into its first order logic.

- a. $\forall x: \text{food}(x) \rightarrow \text{likes}(\text{John}, x)$
 - b. $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
 - c. $\forall x \forall y: \text{eats}(x, y) \wedge \neg \text{killed}(x) \rightarrow \text{food}(y)$
 - d. $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$.
 - e. $\forall x : \text{eats}(\text{Anil}, x) \rightarrow \text{eats}(\text{Harry}, x)$
 - f. $\forall x: \neg \text{killed}(x) \rightarrow \text{alive}(x)$
 - g. $\forall x: \text{alive}(x) \rightarrow \neg \text{killed}(x)$
 - h. $\text{likes}(\text{John}, \text{Peanuts})$
- } **added predicates.**

Step-2: Conversion of FOL into CNF

In First order logic resolution, it is required to convert the FOL into CNF as the CNF form makes resolution proofs.

- **Eliminate all implication (\rightarrow) and rewrite**
 - a. $\forall x \neg \text{food}(x) \vee \text{likes}(\text{John}, x)$
 - b. $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
 - c. $\forall x \forall y \neg [\text{eats}(x, y) \wedge \neg \text{killed}(x)] \vee \text{food}(y)$
 - d. $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$
 - e. $\forall x \neg \text{eats}(\text{Anil}, x) \vee \text{eats}(\text{Harry}, x)$
 - f. $\forall x \neg [\neg \text{killed}(x)] \vee \text{alive}(x)$
 - g. $\forall x \neg \text{alive}(x) \vee \neg \text{killed}(x)$
 - h. $\text{likes}(\text{John}, \text{Peanuts})$.
- **Move negation (\neg) inwards and rewrite**
 - a. $\forall x \neg \text{food}(x) \vee \text{likes}(\text{John}, x)$
 - b. $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
 - c. $\forall x \forall y \neg \text{eats}(x, y) \vee \text{killed}(x) \vee \text{food}(y)$
 - d. $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$
 - e. $\forall x \neg \text{eats}(\text{Anil}, x) \vee \text{eats}(\text{Harry}, x)$
 - f. $\forall x \neg \text{killed}(x) \vee \text{alive}(x)$
 - g. $\forall x \neg \text{alive}(x) \vee \neg \text{killed}(x)$
 - h. $\text{likes}(\text{John}, \text{Peanuts})$.
- **Rename variables or standardize variables**
 - a. $\forall x \neg \text{food}(x) \vee \text{likes}(\text{John}, x)$

- b. $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
- c. $\forall y \forall z \neg \text{eats}(y, z) \vee \text{killed}(y) \vee \text{food}(z)$
- d. $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$
- e. $\forall w \neg \text{eats}(\text{Anil}, w) \vee \text{eats}(\text{Harry}, w)$
- f. $\forall g \neg \text{killed}(g) \vee \text{alive}(g)$
- g. $\forall k \neg \text{alive}(k) \vee \neg \text{killed}(k)$
- h. $\text{likes}(\text{John}, \text{Peanuts})$.

- **Eliminate existential instantiation quantifier by elimination.**

In this step, we will eliminate existential quantifier \exists , and this process is known as **Skolemization**. But in this example problem since there is no existential quantifier so all the statements will remain the same in this step.

- **Drop Universal quantifiers.**

In this step we will drop all universal quantifiers since all the statements are not implicitly quantified so we don't need it.

- a. $\neg \text{food}(x) \vee \text{likes}(\text{John}, x)$
- b. $\text{food}(\text{Apple})$
- c. $\text{food}(\text{vegetables})$
- d. $\neg \text{eats}(y, z) \vee \text{killed}(y) \vee \text{food}(z)$
- e. $\text{eats}(\text{Anil}, \text{Peanuts})$
- f. $\text{alive}(\text{Anil})$
- g. $\neg \text{eats}(\text{Anil}, w) \vee \text{eats}(\text{Harry}, w)$
- h. $\text{killed}(g) \vee \text{alive}(g)$
- i. $\neg \text{alive}(k) \vee \neg \text{killed}(k)$
- j. $\text{likes}(\text{John}, \text{Peanuts})$.

Note: Statements " $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$ " and " $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$ " can be written in two separate statements.

- **Distribute conjunction \wedge over disjunction \vee .**

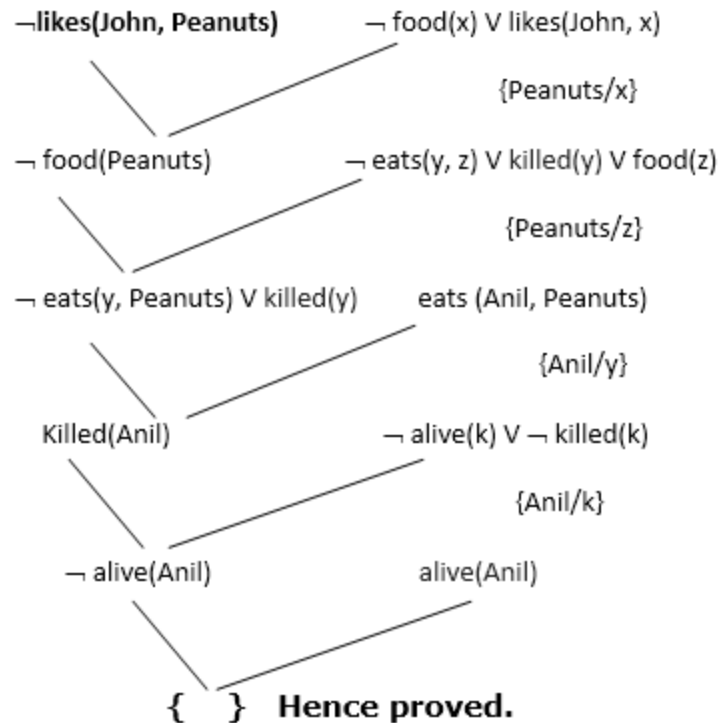
This step will not make any change in this problem.

Step-3: Negate the statement to be proved

In this statement, we will apply negation to the conclusion statements, which will be written as $\neg \text{likes}(\text{John}, \text{Peanuts})$

Step-4: Draw Resolution graph:

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



Hence the negation of the conclusion has been proved as a complete contradiction with the given set of statements.

Explanation of Resolution graph:

- In the first step of resolution graph, $\neg \text{likes}(\text{John}, \text{Peanuts})$, and $\text{likes}(\text{John}, x)$ get resolved(canceled) by substitution of $\{\text{Peanuts}/x\}$, and we are left with $\neg \text{food}(\text{Peanuts})$
- In the second step of the resolution graph, $\neg \text{food}(\text{Peanuts})$, and $\text{food}(z)$ get resolved (canceled) by substitution of $\{\text{Peanuts}/z\}$, and we are left with $\neg \text{eats}(y, \text{Peanuts}) \vee \text{killed}(y)$.
- In the third step of the resolution graph, $\neg \text{eats}(y, \text{Peanuts})$ and $\text{eats}(\text{Anil}, \text{Peanuts})$ get resolved by substitution $\{\text{Anil}/y\}$, and we are left with $\text{Killed}(\text{Anil})$.
- In the fourth step of the resolution graph, $\text{Killed}(\text{Anil})$ and $\neg \text{killed}(k)$ get resolved by substitution $\{\text{Anil}/k\}$, and we are left with $\neg \text{alive}(\text{Anil})$.
- In the last step of the resolution graph $\neg \text{alive}(\text{Anil})$ and $\text{alive}(\text{Anil})$ get resolved.

What is knowledge representation?

Humans are best at understanding, reasoning, and interpreting knowledge. Humans know 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.** Hence we can describe Knowledge representation as following:

- Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which is concerned with AI agents thinking and how thinking contributes to intelligent behavior of agents.
- It is responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve complex real world problems such as diagnosing a medical condition or communicating with humans in natural language.
- 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:

Following are the kind of knowledge which needs to be represented in AI systems:

- **Object:** All the facts about objects in our world domain. E.g Guitars contain strings, trumpets are brass instruments.
- **Events:** Events are the actions which occur in our world.
- **Performance:** It describes behavior 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).

Knowledge: Knowledge is awareness or familiarity gained by experiences of facts, data, and situations.

Ontological engineering

There are two main approaches to supporting Artificial Intelligence (AI). On the one hand, through machine learning where AI systems learn from examples. We hear a lot about machine learning, as represented by IBM's Watson. Machine learning-based systems use statistical classification of patterns to compare what they learn from the training set with new data to see if they fit the pattern. For example, it is often used to predict fraud that can be detected by analyzing patterns in the data and comparing them to patterns that are known to be related to fraud.

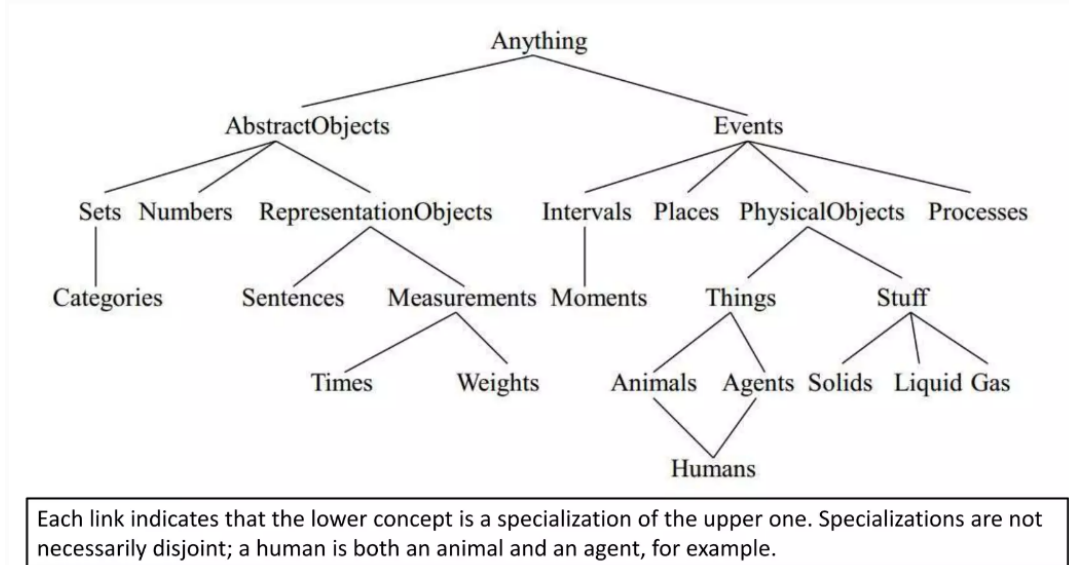
Another approach to supporting AI is ontology. Ontology is a set of subject or domain concepts and categories that show their properties and the relationships between them.

Ontology refers to organizing everything in the world into a hierarchy of categories. Representing abstract concepts such as Actions, Time, Physical Objects, and Beliefs is called Ontological Engineering.

This approach creates a knowledge base that systematizes information, and then develops an ontology that reflects the associations between different data items. Ontology-based AI allows systems to make inferences based on content and relationships, emulating human performance. Ontology-based AI can provide targeted results and does not require the use of training sets to function.

The knowledge model runs along a continuum starting at the simplest level. There, a controlled vocabulary is developed that encourages the use of the same word for specific meanings. The next step is to be able to identify terms related to a single concept. The next level is a taxonomy that defines a hierarchy of parent-child relationships. Parent-child relationships can be a specialization of one product category or an item classified as part of another product category, such as a product, engines as part of a car. These are often used as a navigation structure for websites to move users from one information to another. Ontologies represent relationships between multiple classifications. Finally, the knowledge graph can be used to capture a specific instance of a relationship, such as a specific sales transaction between entities, but ontology is common. Types of relationships in the model includes

- Equivalence is used in thesaurus and does not necessarily refer to synonyms. For example, transparency and opacity both refer to the same concept, but with different explanations.
- Hierarchical relationships are used in classification methods to classify elements or concepts, and related terms are used for ontological concepts or entities.
- Related terms are contextual and audience-specific and are used to relate multiple classifications.



Ontologies and classifications share several elements, but ontology provides a richer set of information. The taxonomy is a tree structure, and an element or concept can have only one parent. Ontologies allow objects like smartphones to inherit functionality from multiple parents. This quality is useful because real-world objects and concepts are not just one, but interrelated in many ways.

Ontology real world example

Ontologies can be applied to situations that evaluate customer behavior. The Cleveland Museum of Art wanted to understand the patterns of visitor preferences and interactions with the museum's collections. To do this, they had to identify the characteristics of the collection, their themes, locations, and specific points of interaction for the visitor. The term for location was developed because it was needed to establish a correlation between the visitor's reaction and the exhibit. Visitors agreed that the museum would record what they said. Ontologies were developed to connect geospatial data with behavioral analytics based on highly specific content. The museum was able to better understand the tastes of the visitors and provide a better experience.

Ontology and digital transformation

Hundreds or thousands of AI projects are required to achieve complete digital transformation, so it's important to set the stage for scaling up. Instead of having a series of separate projects, you need to develop an ineffable framework. In addition, existing ontology allows you to make changes to your data in one place and propagate them through existing relationships. For example, if the price of a service changes, the system does not need to be recorded in multiple applications. Data can be modified once and transferred to all. Robust information architectures are needed to support the advanced solutions that emerge as organizations drive digital transformation. Ontology provides a reusable and adaptable structure for organizations looking to drive AI initiatives. The more detailed the ontology, the more meaningful the user will get.

CATEGORIES AND OBJECTS

The organization of objects into categories is a vital part of knowledge representation. Although interaction with the world takes place at the level of individual objects, much reasoning takes place at the level of categories.

There are two choices for representing categories in first-order logic: predicates and objects. That is, we can use the predicate Basketball(b), or we can reify¹ the category as an object, Basketballs. We could then say Member(b, Basketballs), which we will abbreviate as $b \in \text{Basketballs}$, to say that b is a member of the category of basketballs. We say Subset(Basketballs, Balls), abbreviated as $\text{Basketballs} \subset \text{Balls}$, to say that Basketballs is a subcategory of Balls. We will use subcategory, subclass, and subset interchangeably.

Categories serve to organize and simplify the knowledge base through inheritance. If we say that all instances of the category Food are edible, and if we assert that Fruit is a subclass of Food and Apples is a subclass of Fruit, then we can infer that every apple is edible. We say that the individual apples inherit the property of edibility, in this case from their membership in the Food category.

First-order logic makes it easy to state facts about categories, either by relating objects to categories or by quantifying over their members. Here are some types of facts, with examples of each:

- An object is a member of a category. $\text{BB9} \in \text{Basketballs}$
- A category is a subclass of another category. $\text{Basketballs} \subset \text{Balls}$
- All members of a category have some properties. $(x \in \text{Basketballs}) \Rightarrow \text{Spherical}(x)$
- Members of a category can be recognized by some properties.

$$\text{Orange}(x) \wedge \text{Round}(x) \wedge \text{Diameter}(x)=9.5 \wedge x \in \text{Balls} \Rightarrow x \in \text{Basketballs}$$

- A category as a whole has some properties. $\text{Dogs} \in \text{DomesticatedSpecies}$

Although subclass and member relations are the most important ones for categories, we also want to be able to state relations between categories that are not subclasses of each other.

For example, if we just say that Males and Females are subclasses of Animals, then we have not said that a male cannot be a female. We say that two or more categories are disjoint if they have no members in common. And even if we know that males and females are disjoint, we will not know that an animal that is not a male must be a female, unless we say that males and females constitute an exhaustive decomposition of the animals. A disjoint exhaustive decomposition is known as a partition. The following examples illustrate these three concepts:

$$\begin{aligned} & \text{Disjoint}(\{\text{Animals}, \text{Vegetables}\}) \\ & \text{ExhaustiveDecomposition}(\{\text{Americans}, \text{Canadians}, \text{Mexicans}\}, \\ & \quad \text{NorthAmericans}) \\ & \text{Partition}(\{\text{Males}, \text{Females}\}, \text{Animals}) . \end{aligned}$$

(Note that the ExhaustiveDecomposition of NorthAmericans is not a Partition, because some people have dual citizenship.)

The three predicates are defined as follows:

$$\begin{aligned} \text{Disjoint}(s) & \Leftrightarrow (\forall c1, c2 \ c1 \in s \wedge c2 \in s \wedge c1 \neq c2 \Rightarrow \text{Intersection}(c1, c2) = \{\}) \\ \text{ExhaustiveDecomposition}(s, c) & \Leftrightarrow (\forall i \ i \in c \Leftrightarrow \exists c2 \ c2 \in s \wedge i \in c2) \\ \text{Partition}(s, c) & \Leftrightarrow \text{Disjoint}(s) \wedge \text{ExhaustiveDecomposition}(s, c) . \end{aligned}$$

Categories can also be defined by providing necessary and sufficient conditions for membership. For example, a bachelor is an unmarried adult male:

$$x \in \text{Bachelors} \Leftrightarrow \text{Unmarried}(x) \wedge x \in \text{Adults} \wedge x \in \text{Males} .$$

Physical composition

The idea that one object can be part of another is a familiar one. One's nose is part of one's head, Romania is part of Europe, and this chapter is part of this book. We use the general PartOf relation to say that one thing is part of another. Objects can be grouped into PartOf hierarchies, reminiscent of the Subset hierarchy:

$$\begin{aligned} &\text{PartOf}(\text{Bucharest}, \text{Romania}) \\ &\text{PartOf}(\text{Romania}, \text{EasternEurope}) \\ &\text{PartOf}(\text{EasternEurope}, \text{Europe}) \\ &\text{PartOf}(\text{Europe}, \text{Earth}) . \end{aligned}$$

The PartOf relation is transitive and reflexive; that is,

$$\begin{aligned} &\text{PartOf}(x, y) \wedge \text{PartOf}(y, z) \Rightarrow \text{PartOf}(x, z) . \\ &\text{PartOf}(x, x) . \end{aligned}$$

Therefore, we can conclude PartOf(Bucharest, Earth). Categories of composite objects are often characterized by structural relations among parts. For example, a biped has two legs attached to a body:

$$\begin{aligned} \text{Biped}(a) \Rightarrow &\exists l_1, l_2, b \text{ Leg}(l_1) \wedge \text{Leg}(l_2) \wedge \text{Body}(b) \wedge \\ &\text{PartOf}(l_1, a) \wedge \text{PartOf}(l_2, a) \wedge \text{PartOf}(b, a) \wedge \\ &\text{Attached}(l_1, b) \wedge \text{Attached}(l_2, b) \wedge \\ &l_1 \neq l_2 \wedge [\forall l_3 \text{ Leg}(l_3) \wedge \text{PartOf}(l_3, a) \Rightarrow (l_3 = l_1 \vee l_3 = l_2)] . \end{aligned}$$

The notation for “exactly two” is a little awkward; we are forced to say that there are two legs, that they are not the same, and that if anyone proposes a third leg, it must be the same as one of the other two.

We can define a PartPartition relation analogous to the Partition relation for categories. An object is composed of the parts in its PartPartition and can be viewed as deriving some properties from those parts. For example, the mass of a composite object is the sum of the masses of the parts. Notice that this is not the case with categories, which have no mass, even though their elements might.

It is also useful to define composite objects with definite parts but no particular structure. For example, we might want to say “The apples in this bag weigh two pounds.” The temptation would be to ascribe this weight to the set of apples in the bag, but this would be a mistake because the set is an abstract mathematical concept that has elements but does not BUNCH have weight. Instead, we need a new concept, which we will call a bunch.

$$\begin{aligned} &\text{For example, if the apples are Apple1, Apple2, and Apple3, then} \\ &\text{BunchOf}(\{\text{Apple1}, \text{Apple2}, \text{Apple3}\}) \end{aligned}$$


denotes the composite object with the three apples as parts (not elements). We can then use the bunch as a normal, albeit unstructured, object. Notice that $\text{BunchOf}(\{x\}) = x$. Furthermore, $\text{BunchOf}(\text{Apples})$ is the composite object consisting of all apples—not to be confused with Apples, the category or set of all apples. We can define BunchOf in terms of the PartOf relation. Obviously, each element of s is part of E

Furthermore, $\text{BunchOf}(s)$ is the smallest object satisfying this condition. In other words, $\text{BunchOf}(s)$ must be part of any object that has all the elements of s as parts:

$$\forall y [\forall x x \in s \Rightarrow \text{PartOf}(x, y)] \Rightarrow \text{PartOf}(\text{BunchOf}(s), y) .$$

These axioms are an example of a general technique called logical minimization, which means defining an object as the smallest one satisfying certain conditions.

Measurements

In both scientific and common sense theories of the world, objects have height, mass, cost, and so on. The values that we assign for these properties are called measures. Ordinary quantitative measures are quite easy to represent. We imagine that the universe includes abstract “measure objects,” such as the length that is the length of this line segment: . We can call this length 1.5 inches or 3.81 centimeters. Thus, the same length has different names in our language. We represent the length with a unit function that takes a number as an argument.

$$\text{Length}(L1) = \text{Inches}(1.5) = \text{Centimeters}(3.81) .$$

Conversion between units is done by equating multiples of one unit to another:

$$\text{Centimeters}(2.54 \times d) = \text{Inches}(d) .$$

Similar axioms can be written for pounds and kilograms, seconds and days, and dollars and cents. Measures can be used to describe objects as follows:

$$\text{Diameter}(\text{Basketball } 12) = \text{Inches}(9.5) .$$

$$\text{ListPrice}(\text{Basketball } 12) = \$ (19) .$$

$$d \in \text{Days} \Rightarrow \text{Duration}(d) = \text{Hours}(24) .$$

Note that \$(1) is not a dollar bill! One can have two dollar bills, but there is only one object named \$(1). Note also that, while Inches(0) and Centimeters(0) refer to the same zero length, they are not identical to other zero measures, such as Seconds(0).

Simple, quantitative measures are easy to represent. Other measures present more of a problem, because they have no agreed scale of values. Exercises have difficulty, desserts have deliciousness, and poems have beauty, yet numbers cannot be assigned to these qualities. One might, in a moment of pure accountancy, dismiss such properties as useless for the purpose of logical reasoning; or, still worse, attempt to impose a numerical scale on beauty. This would be a grave mistake, because it is unnecessary. The most important aspect of measures is not the particular numerical values, but the fact that measures can be ordered.

Although measures are not numbers, we can still compare them, using an ordering symbol such as >. For example, we might well believe that Norvig’s exercises are tougher than Russell’s, and that one scores less on tougher exercises:

$$e1 \in \text{Exercises} \wedge e2 \in \text{Exercises} \wedge \text{Wrote}(\text{Norvig}, e1) \wedge \text{Wrote}(\text{Russell}, e2) \Rightarrow$$

$$\text{Difficulty}(e1) > \text{Difficulty}(e2) .$$

$$e1 \in \text{Exercises} \wedge e2 \in \text{Exercises} \wedge \text{Difficulty}(e1) > \text{Difficulty}(e2) \Rightarrow$$

$$\text{ExpectedScore}(e1) < \text{ExpectedScore}(e2) .$$

This is enough to allow one to decide which exercises to do, even though no numerical values for difficulty were ever used. (One does, however, have to discover who wrote which exercises.) These sorts of monotonic relationships among measures form the basis for the field of qualitative physics, a subfield of AI that investigates how to reason about physical systems without plunging into detailed equations and numerical simulations. Qualitative physics is discussed in the historical notes section.

EVENTS

Situation calculus is limited in its applicability: it was designed to describe a world in which actions are discrete, instantaneous, and happen one at a time. Consider a continuous action, such as filling a bathtub. Situation calculus can say that the tub is empty before the action and full when the action is done, but it can't talk about what happens during the action. It also can't describe two actions happening at the same time—such as brushing one's teeth while waiting for the tub to fill. To handle such cases we introduce an alternative formalism known as event calculus, which is based on points of time rather than on situations.

Event calculus reifies fluents and events. The fluent $At(Shankar, Berkeley)$ is an object that refers to the fact of Shankar being in Berkeley, but does not by itself say anything about whether it is true. To assert that a fluency is actually true at some point in time we use the predicate T , as in $T(At(Shankar, Berkeley), t)$.

Events are described as instances of event categories.⁴ The event $E1$ of Shankar flying from San Francisco to Washington, D.C. is described as

$$E1 \in \text{Flyings} \wedge \text{Flyer}(E1, \text{Shankar}) \wedge \text{Origin}(E1, \text{SF}) \wedge \text{Destination}(E1, \text{DC}).$$

If this is too verbose, we can define an alternative three-argument version of the category of flying events and say

$$E1 \in \text{Flyings}(\text{Shankar}, \text{SF}, \text{DC}).$$

We then use $\text{Happens}(E1, i)$ to say that the event $E1$ took place over the time interval i , and we say the same thing in functional form with $\text{Extent}(E1) = i$. We represent time intervals by a (start, end) pair of times; that is, $i = (t1, t2)$ is the time interval that starts at $t1$ and ends at $t2$. The complete set of predicates for one version of the event calculus is

$$\begin{aligned} T(f, t) & \text{Fluent } f \text{ is true at time } t \\ \text{Happens}(e, i) & \text{Event } e \text{ happens over the time interval } i \\ \text{Initiates}(e, f, t) & \text{Event } e \text{ causes fluent } f \text{ to start to hold at time } t \\ \text{Terminates}(e, f, t) & \text{Event } e \text{ causes fluent } f \text{ to cease to hold at time } t \\ \text{Clipped}(f, i) & \text{Fluent } f \text{ ceases to be true at some point during time interval } i \\ \text{Restored}(f, i) & \text{Fluent } f \text{ becomes true sometime during time interval } i \end{aligned}$$

We assume a distinguished event, Start , that describes the initial state by saying which fluents are initiated or terminated at the start time. We define T by saying that a fluent holds at a point in time if the fluent was initiated by an event at some time in the past and was not made false (clipped) by an intervening event. A fluent does not hold if it was terminated by an event and not made true (restored) by another event. Formally, the axioms are:

$$\begin{aligned} \text{Happens}(e, (t1, t2)) \wedge \text{Initiates}(e, f, t1) \wedge \neg \text{Clipped}(f, (t1, t)) \wedge t1 < t &\Rightarrow T(f, t) \\ \text{Happens}(e, (t1, t2)) \wedge \text{Terminates}(e, f, t1) \wedge \neg \text{Restored}(f, (t1, t)) \wedge t1 < t &\Rightarrow \neg T(f, t) \end{aligned}$$

where Clipped and Restored are defined by

$$\begin{aligned} \text{Clipped}(f, (t1, t2)) &\Leftrightarrow \exists e, t, t3 \text{ Happens}(e, (t, t3)) \wedge t1 \leq t < t2 \wedge \text{Terminates}(e, f, t) \\ \text{Restored}(f, (t1, t2)) &\Leftrightarrow \exists e, t, t3 \text{ Happens}(e, (t, t3)) \wedge t1 \leq t < t2 \wedge \text{Initiates}(e, f, t) \end{aligned}$$

It is convenient to extend T to work over intervals as well as time points; a fluent holds over an interval if it holds on every point within the interval:

$$T(f, (t1, t2)) \Leftrightarrow [\forall t (t1 \leq t < t2) \Rightarrow T(f, t)]$$

Fluents and actions are defined with domain-specific axioms that are similar to successor state axioms. For example, we can say that the only way a wumpus-world agent gets an arrow is at the start, and the only way to use up an arrow is to shoot it:

$$\begin{aligned}\text{Initiates}(e, \text{HaveArrow}(a), t) &\Leftrightarrow e = \text{Start} \\ \text{Terminates}(e, \text{HaveArrow}(a), t) &\Leftrightarrow e \in \text{Shootings}(a)\end{aligned}$$

By reifying events we make it possible to add any amount of arbitrary information about them. For example, we can say that Shankar's flight was bumpy with $\text{Bumpy}(E1)$. In an ontology where events are n -ary predicates, there would be no way to add extra information like this; moving to an $n + 1$ -ary predicate isn't a scalable solution.

We can extend event calculus to make it possible to represent simultaneous events (such as two people being necessary to ride a seesaw), exogenous events (such as the wind blowing and changing the location of an object), continuous events (such as the level of water in the bathtub continuously rising) and other complications.

Mental Events and Mental Objects

A mental event is any event that happens within the mind of a conscious individual. Examples include thoughts, feelings, decisions, dreams, and realizations. The mental object is the range of what one has perceived, discovered, or learned. The term is related to the cognition of the human brain. The mental object towards the same stimuli or in the same surrounding differs variedly. Every brain perceives and learns the information in a very different way.

For example, suppose Alice asks "what is the square root of 1764" and Bob replies "I don't know." If Alice insists "think harder," Bob should realize that with some more thought, this question can in fact be answered. On the other hand, if the question were "Is your mother sitting down right now?" then Bob should realize that thinking harder is unlikely to help. Knowledge about the knowledge of other agents is also important; Bob should realize that his mother knows whether she is sitting or not, and that asking her would be a way to find out.

What we need is a model of the mental objects that are in someone's head (or something's knowledge base) and of the mental processes that manipulate those mental objects. The model does not have to be detailed. We do not have to be able to predict how many milliseconds it will take for a particular agent to make a deduction. We will be happy just to be able to conclude that mother knows whether or not she is sitting.

We begin with the propositional attitudes that an agent can have toward mental objects: attitudes such as Believes, Knows, Wants, Intends, and Informs. The difficulty is that these attitudes do not behave like "normal" predicates. For example, suppose we try to assert that Lois knows that Superman can fly:

$$\text{Knows}(\text{Lois}, \text{CanFly}(\text{Superman})).$$

One minor issue with this is that we normally think of $\text{CanFly}(\text{Superman})$ as a sentence, but here it appears as a term. That issue can be patched up just by reifying $\text{CanFly}(\text{Superman})$; making it a fluent. A more serious problem is that, if it is true that Superman is Clark Kent, then we must conclude that Lois knows that Clark can fly:

$$\begin{aligned}(\text{Superman} = \text{Clark}) \wedge \text{Knows}(\text{Lois}, \text{CanFly}(\text{Superman})) \\ \models \text{Knows}(\text{Lois}, \text{CanFly}(\text{Clark})).\end{aligned}$$

This is a consequence of the fact that equality reasoning is built into logic. Normally that is a good thing; if our agent knows that $2+2=4$ and $4 < 5$, then we want our agent to know that $2+2 < 5$. This property is called referential transparency; it doesn't matter what term a logic uses to refer to an object, what matters is the object that the term names. But for propositional attitudes like believes and knows, we would like to have referential opacity the terms used do matter, because not all agents know which terms are co-referential.

Modal logic is designed to address this problem. Regular logic is concerned with a single modality, the modality of truth, allowing us to express “P is true.” Modal logic includes special modal operators that take sentences (rather than terms) as arguments. For example, “A knows P” is represented with the notation KAP , where K is the modal operator for knowledge. It takes two arguments, an agent (written as the subscript) and a sentence. The syntax of modal logic is the same as first-order logic, except that sentences can also be formed with modal operators.

The semantics of modal logic is more complicated. In first-order logic a model contains a set of objects and an interpretation that maps each name to the appropriate object, relation, or function. In modal logic we want to be able to consider both the possibility that Superman's secret identity is Clark and that it isn't. Therefore, we will need a more complicated model, one that consists of a collection of possible worlds rather than just one true world. The worlds are connected in a graph by accessibility relations, one relation for each modal operator. We say that world w_1 is accessible from world w_0 with respect to the modal operator KA if everything in w_1 is consistent with what A knows in w_0 , and we write this as $\text{Acc}(KA, w_0, w_1)$. In diagrams such as Figure 12.4 we show accessibility as an arrow between possible worlds. As an example, in the real world, Bucharest is the capital of Romania, but for an agent that did not know that, other possible worlds are accessible, including ones where the capital of Romania is Sibiu or Sofia. Presumably a world where $2+2=5$ would not be accessible to any agent.

In general, a knowledge atom KAP is true in world w if and only if P is true in every world accessible from w . The truth of more complex sentences is derived by recursive application of this rule and the normal rules of first-order logic. That means that modal logic can be used to reason about nested knowledge sentences: what one agent knows about another agent's knowledge. For example, we can say that, even though Lois doesn't know whether Superman's secret identity is Clark Kent, she does know that Clark knows:

$$K\text{Lois} [K\text{Clark Identity}(\text{Superman}, \text{Clark}) \vee K\text{Clark}\neg\text{Identity}(\text{Superman}, \text{Clark})]$$

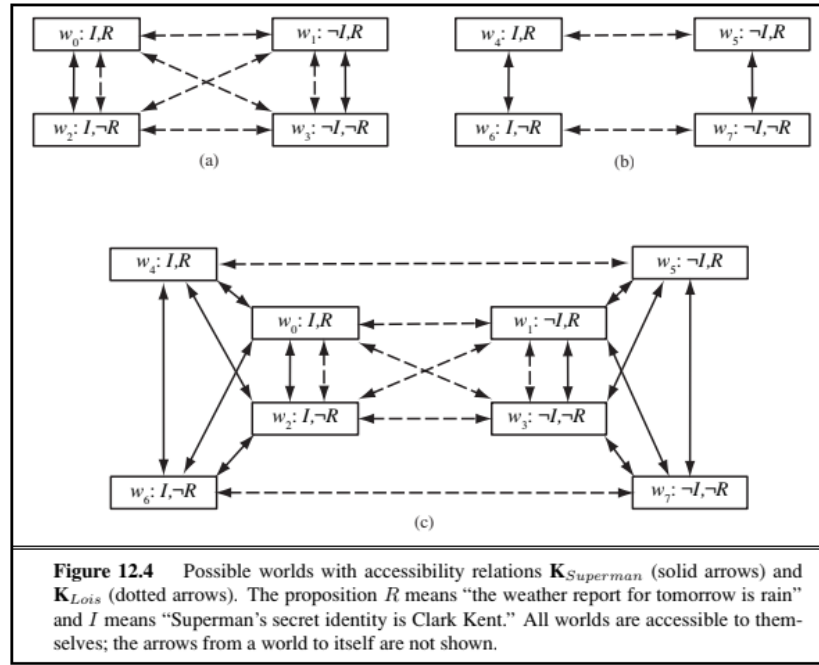
Figure 12.4 shows some possible worlds for this domain, with accessibility relations for Lois and Superman.

In the diagram, it is common knowledge that Superman knows his own identity, and neither he nor Lois has seen the weather report. So in w_0 the worlds w_0 and w_2 are accessible to Superman; maybe rain is predicted, maybe not. For Lois all four worlds are accessible from each other; she doesn't know anything about the report or if Clark is Superman. But she does know that Superman knows whether he is Clark, because in every world that is accessible to Lois, either Superman knows I , or he knows $\neg I$. Lois does not know which is the case, but either way she knows Superman knows.

In the TOP-RIGHT diagram it is common knowledge that Lois has seen the weather report. So in w_4 she knows rain is predicted and in w_6 she knows rain is not predicted. Superman does not know the report, but he knows that Lois knows, because in every world that is accessible to him, either she knows R or she knows $\neg R$. In the BOTTOM diagram we represent the scenario where it is common knowledge that

Superman knows his identity, and Lois might or might not have seen the weather report. We represent this by combining the two top scenarios, and adding arrows to show that Superman does not know which scenario actually holds. Lois does know, so we don't need to add any arrows for her. In w_0 Superman still knows I but not R , and now he does not know whether Lois knows R . From what Superman knows, he might be in w_0 or w_2 , in which case Lois does not know whether R is true, or he could be in w_4 , in which case she knows R , or

w6, in which case she knows $\neg R$.



There are an infinite number of possible worlds, so the trick is to introduce just the ones you need to represent what you are trying to model. A new possible world is needed to talk about different possible facts (e.g., rain is predicted or not), or to talk about different states of knowledge (e.g., does Lois know that rain is predicted). That means two possible worlds, such as w4 and w0 in Figure 12.4, might have the same base facts about the world, but differ in their accessibility relations, and therefore in facts about knowledge.

Modal logic solves some tricky issues with the interplay of quantifiers and knowledge. The English sentence “Bond knows that someone is a spy” is ambiguous. The first reading is that there is a particular someone who Bond knows is a spy; we can write this as

$$\exists x \text{ KBondSpy}(x),$$

which in modal logic means that there is an x that, in all accessible worlds, Bond knows to be a spy. The second reading is that Bond just knows that there is at least one spy:

$$\text{KBond} \exists x \text{ Spy}(x).$$

The modal logic interpretation is that in each accessible world there is an x that is a spy, but it need not be the same x in each world.

Now that we have a modal operator for knowledge, we can write axioms for it. First, we can say that agents are able to draw deductions; if an agent knows P and knows that P implies Q , then the agent knows Q :

$$(\text{Ka}P \wedge \text{Ka}(P \Rightarrow Q)) \Rightarrow \text{Ka}Q .$$

From this (and a few other rules about logical identities) we can establish that $\text{KA}(P \vee \neg P)$ is a tautology; every agent knows every proposition P is either true or false. On the other hand, $(\text{KAP}) \vee (\text{KA}\neg P)$ is not a tautology; in general, there will be lots of propositions that an agent does not know to be true and does not know to be false.

It is said (going back to Plato) that knowledge is justified true belief. That is, if it is true, if you believe it, and if you have an unassailable good reason, then you know it. That means that if you know something, it must be true, and we have the axiom:

$$KaP \Rightarrow P .$$

Furthermore, logical agents should be able to introspect on their own knowledge. If they know something, then they know that they know it:

$$KaP \Rightarrow Ka(KaP) .$$

We can define similar axioms for belief (often denoted by B) and other modalities. However, one problem with the modal logic approach is that it assumes logical omniscience on the part of agents. That is, if an agent knows a set of axioms, then it knows all consequences of those axioms. This is on shaky ground even for the somewhat abstract notion of knowledge, but it seems even worse for belief, because belief has more connotation of referring to things that are physically represented in the agent, not just potentially derivable. There have been attempts to define a form of limited rationality for agents; to say that agents believe those assertions that can be derived with the application of no more than k reasoning steps, or no more than s seconds of computation. These attempts have been generally unsatisfactory.

REASONING SYSTEMS FOR CATEGORIES

Semantic networks are alternatives to predicate logic for knowledge representation. In Semantic networks, we can represent our knowledge in the form of graphical networks. This network consists of nodes representing objects and arcs which describe the relationship between those objects. Semantic networks can categorize the object in different forms and can also link those objects. Semantic networks are easy to understand and can be easily extended.

There are many variants of semantic networks, but all are capable of representing individual objects, categories of objects, and relations among objects. A typical graphical notation displays object or category names in ovals or boxes, and connects them with labeled links.

For example, Figure 12.5 has a MemberOf link between Mary and FemalePersons, corresponding to the logical assertion $Mary \in FemalePersons$; similarly, the SisterOf link between Mary and John corresponds to the assertion $SisterOf(Mary, John)$. We can connect categories using SubsetOf links, and so on. It is such fun drawing bubbles and arrows that one can get carried away. For example, we know that persons have female persons as mothers, so can we draw a HasMother link from Persons to FemalePersons? The answer is no, because HasMother is a relation between a person and his or her mother, and categories do not have mothers.

For this reason, we have used a special notation—the double-boxed link—in Figure 12.5.

This link asserts that

$$\forall x x \in Persons \Rightarrow [\forall y HasMother(x, y) \Rightarrow y \in FemalePersons] .$$

We might also want to assert that persons have two legs—that is,

$$\forall x x \in Persons \Rightarrow Legs(x, 2) .$$

As before, we need to be careful not to assert that a category has legs; the single-boxed link in Figure 12.5 is used to assert properties of every member of a category.

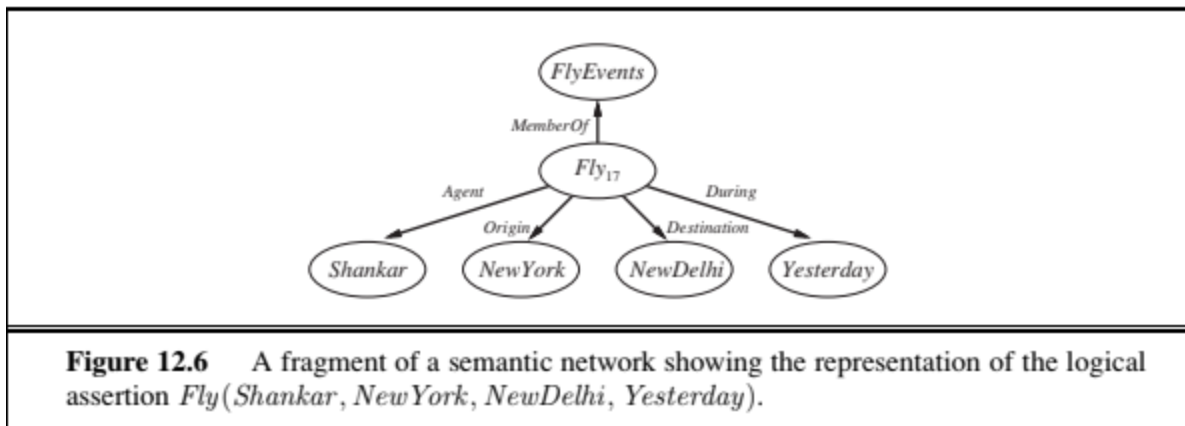
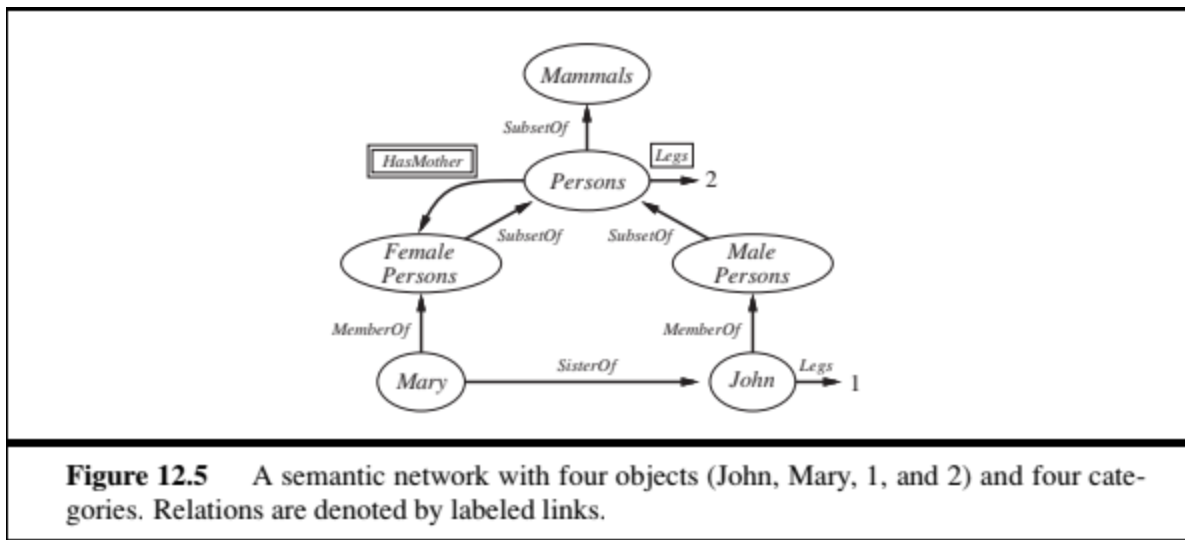
The semantic network notation makes it convenient to perform inheritance reasoning of the kind introduced in

Section 12.2. For example, by virtue of being a person, Mary inherits the property of having two legs. Thus, to find out how many legs Mary has, the inheritance algorithm follows the MemberOf link from Mary to the category she belongs to, and then follows SubsetOf links up the hierarchy until it finds a category for which there is a boxed Legs link—in this case, the Persons category. The simplicity and efficiency of this inference mechanism, compared with logical theorem proving, has been one of the main attractions of semantic networks.

Inheritance becomes complicated when an object can belong to more than one category or when a category can be a subset of more than one other category; this is called multiple inheritance. In such cases, the inheritance algorithm might find two or more conflicting values answering the query. For this reason, multiple inheritance is banned in some object-oriented programming (OOP) languages, such as Java, that use inheritance in a class hierarchy. It is usually allowed in semantic networks, but we defer discussion of that until Section 12.6.

The reader might have noticed an obvious drawback of semantic network notation, compared to first-order logic: the fact that links between bubbles represent only binary relations. For example, the sentence Fly(Shankar, NewYork, NewDelhi, Yesterday) cannot be asserted directly in a semantic network. Nonetheless, we can obtain the effect of n-ary assertions by reifying the proposition itself as an event belonging to an appropriate event category. Figure 12.6 shows the semantic network structure for this particular event. Notice that the restriction to binary relations forces the creation of a rich ontology of reified concepts.

Reification of propositions makes it possible to represent every ground, function-free atomic sentence of first-order logic in the semantic network notation. Certain kinds of universally quantified sentences can be asserted using inverse links and the singly boxed and doubly boxed arrows applied to categories, but that still leaves us a long way short of full first-order logic. Negation, disjunction, nested function symbols, and existential quantification are all missing. Now it is possible to extend the notation to make it equivalent to first-order logic—as in Peirce’s existential graphs—but doing so negates one of the main advantages of semantic networks, which is the simplicity and transparency of the inference processes. Designers can build a large network and still have a good idea about what queries will be efficient, because (a) it is easy to visualize the steps that the inference procedure will go through and (b) in some cases the query language is so simple that difficult queries cannot be posed. In cases where the expressive power proves to be too limiting, many semantic network systems provide for procedural attachment to fill in the gaps. Procedural attachment is a technique whereby a query about (or sometimes an assertion of) a certain relation results in a call to a special procedure designed for that relation rather than a general inference algorithm.



One of the most important aspects of semantic networks is their ability to represent default values for categories. Examining Figure 12.5 carefully, one notices that John has one leg, despite the fact that he is a person and all persons have two legs. In a strictly logical KB, this would be a contradiction, but in a semantic network, the assertion that all persons have two legs has only default status; that is, a person is assumed to have two legs unless this is contradicted by more specific information. The default semantics is enforced naturally by the inheritance algorithm, because it follows links upwards from the object itself (John in this case) and stops as soon as it finds a value. We say that the default is overridden by the more specific value. Notice that we could also override the default number of legs by creating a category of OneLeggedPersons, a subset of Persons of which John is a member.

We can retain a strictly logical semantics for the network if we say that the Legs assertion for Persons includes an exception for John:

$$\forall x \, x \in \text{Persons} \wedge x \neq \text{John} \Rightarrow \text{Legs}(x, 2) .$$

For a fixed network, this is semantically adequate but will be much less concise than the network notation itself if there are lots of exceptions. For a network that will be updated with more assertions, however, such an approach fails—we really want to say that any persons as yet unknown with one leg are exceptions too. Section 12.6 goes into more depth on this issue and on default reasoning in general.

This representation consist of mainly two types of relations:

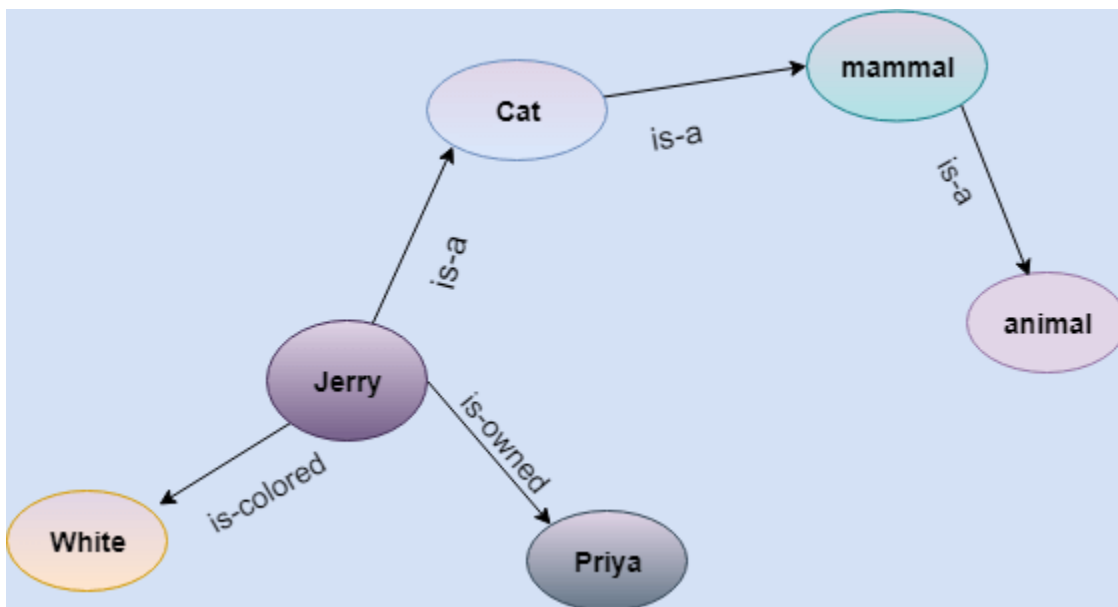
- a. IS-A relation (Inheritance)

b. Kind-of-relation

Example: Following are some statements which we need to represent in the form of nodes and arcs.

Statements:

- Jerry is a cat.
- Jerry is a mammal
- Jerry is owned by Priya.
- Jerry is brown colored.
- All Mammals are animals.



In the above diagram, we have represented the different types of knowledge in the form of nodes and arcs. Each object is connected with another object by some relation.

Drawbacks in Semantic representation:

- Semantic networks take more computational time at runtime as we need to traverse the complete network tree to answer some questions. It might be possible in the worst case scenario that after traversing the entire tree, we find that the solution does not exist in this network.
- Semantic networks try to model human-like memory (Which has 10¹⁵ neurons and links) to store the information, but in practice, it is not possible to build such a vast semantic network.
- These types of representations are inadequate as they do not have any equivalent quantifier, e.g., for all, for some, none, etc.
- Semantic networks do not have any standard definition for the link names.
- These networks are not intelligent and depend on the creator of the system.

Advantages of Semantic network:

- Semantic networks are a natural representation of knowledge.

2. Semantic networks convey meaning in a transparent manner.
3. These networks are simple and easily understandable.

Description logics

The syntax of first-order logic is designed to make it easy to say things about objects. Description logics are notations that are designed to make it easier to describe definitions and properties of categories. Description logic systems evolved from semantic networks in response to pressure to formalize what the networks mean while retaining the emphasis on taxonomic structure as an organizing principle.

The principal inference tasks for description logics are subsumption (checking if one category is a subset of another by comparing their definitions) and classification (checking whether an object belongs to a category). Some systems also include consistency of a category definition—whether the membership criteria are logically satisfiable.

<i>Concept</i>	→	Thing <i>ConceptName</i>
		And (<i>Concept</i> , ...)
		All (<i>RoleName</i> , <i>Concept</i>)
		AtLeast (<i>Integer</i> , <i>RoleName</i>)
		AtMost (<i>Integer</i> , <i>RoleName</i>)
		Fills (<i>RoleName</i> , <i>IndividualName</i> , ...)
		SameAs (<i>Path</i> , <i>Path</i>)
		OneOf (<i>IndividualName</i> , ...)
<i>Path</i>	→	[<i>RoleName</i> , ...]

Figure 12.7 The syntax of descriptions in a subset of the CLASSIC language.

The CLASSIC language (Borgida et al., 1989) is a typical description logic. The syntax of CLASSIC descriptions is shown in Figure 12.7.6 For example, to say that bachelors are unmarried adult males we would write

Bachelor = And(Unmarried, Adult, Male) .

The equivalent in first-order logic would be

$\text{Bachelor}(x) \Leftrightarrow \text{Unmarried}(x) \wedge \text{Adult}(x) \wedge \text{Male}(x)$.

Notice that the description logic has an algebra of operations on predicates, which of course we can't do in first-order logic. Any description in CLASSIC can be translated into an equivalent first-order sentence, but some descriptions are more straightforward in CLASSIC. For example, to describe the set of men with at least three sons who are all unemployed and married to doctors, and at most two daughters who are all professors in physics or math departments, we would use

And(Man, AtLeast(3, Son), AtMost(2, Daughter),
 All(Son, And(Unemployed, Married, All(Spouse, Doctor))),
 All(Daughter , And(Professor ,Fills(Department,Physics, Math)))) .

We leave it as an exercise to translate this into first-order logic.

Perhaps the most important aspect of description logics is their emphasis on tractability of inference. A problem instance is solved by describing it and then asking if it is subsumed by one of several possible solution categories. In standard first-order logic systems, predicting the solution time is often impossible. It is frequently left to the user to engineer the representation to detour around sets of sentences that seem to be causing the system to take several weeks to solve a problem. The thrust in description logics, on the other hand, is to ensure that subsumption-testing can be solved in time polynomial in the size of the descriptions.

This sounds wonderful in principle, until one realizes that it can only have one of two consequences: either hard problems cannot be stated at all, or they require exponentially large descriptions! However, the tractability results do shed light on what sorts of constructs cause problems and thus help the user to understand how different representations behave. For example, description logics usually lack negation and disjunction. Each forces first order logical systems to go through a potentially exponential case analysis in order to ensure completeness. CLASSIC allows only a limited form of disjunction in the Fills and OneOf constructs, which permit disjunction over explicitly enumerated individuals but not over descriptions. With disjunctive descriptions, nested definitions can lead easily to an exponential number of alternative routes by which one category can subsume another.

REASONING WITH DEFAULT INFORMATION

In the preceding section, we saw a simple example of an assertion with default status: people have two legs. This default can be overridden by more specific information, such as that Long John Silver has one leg. We saw that the inheritance mechanism in semantic networks implements the overriding of defaults in a simple and natural way. In this section, we study defaults more generally, with a view toward understanding the semantics of defaults rather than just providing a procedural mechanism.

Circumscription and default logic

We have seen two examples of reasoning processes that violate the monotonicity property of logic that was proved in Chapter 7.8 In this chapter we saw that a property inherited by all members of a category in a semantic network could be overridden by more specific information for a subcategory. In Section 9.4.5, we saw that under the closed-world assumption, if a proposition α is not mentioned in KB then $KB \models \neg\alpha$, but $KB \wedge \alpha \models \alpha$.

Simple introspection suggests that these failures of monotonicity are widespread in commonsense reasoning. It seems that humans often “jump to conclusions.” For example, when one sees a car parked on the street, one is normally willing to believe that it has four wheels even though only three are visible. Now, probability theory can certainly provide a conclusion that the fourth wheel exists with high probability, yet, for most people, the possibility of the car not having four wheels does not arise unless some new evidence presents itself. Thus, it seems that the four-wheel conclusion is reached by default, in the absence of any reason to doubt it. If new evidence arrives—for example, if one sees the owner carrying a wheel and notices that the car is jacked up—then the conclusion can be retracted. This kind of reasoning is said to exhibit nonmonotonicity, because the set of beliefs does not grow monotonically over time as new evidence arrives. Nonmonotonic logics have been devised with modified notions of truth and entailment in order to capture such behavior. We will look at two such logics that have been studied extensively: circumscription and default logic.

Circumscription can be seen as a more powerful and precise version of the closed world assumption. The idea is to specify particular predicates that are assumed to be “as false as possible”—that is, false for every object except those for which they are known to be true. For example, suppose we want to assert the default rule that birds fly. We would introduce a predicate, say Abnormal 1(x), and write

$$\text{Bird}(x) \wedge \neg\text{Abnormal 1}(x) \Rightarrow \text{Flies}(x) .$$

If we say that Abnormal 1 is to be circumscribed, a circumscriptive reasoner is entitled to assume $\neg \text{Abnormal } 1(x)$ unless Abnormal 1(x) is known to be true. This allows the conclusion Flies(Tweety) to be drawn from the premise Bird(Tweety), but the conclusion no longer holds if Abnormal 1(Tweety) is asserted.

Circumscription can be viewed as an example of a model preference logic. In such logics, a sentence is entailed (with default status) if it is true in all preferred models of the KB, as opposed to the requirement of truth in all models in classical logic. For circumscription, one model is preferred to another if it has fewer abnormal objects.⁹ Let us see how this idea works in the context of multiple inheritance in semantic networks. The standard example for which multiple inheritance is problematic is called the “Nixon diamond.” It arises from the observation that Richard Nixon was both a Quaker (and hence by default a pacifist) and a Republican (and hence by default not a pacifist). We can write this as follows:

$$\begin{aligned} & \text{Republican}(\text{Nixon}) \wedge \text{Quaker}(\text{Nixon}) . \\ & \text{Republican}(x) \wedge \neg \text{Abnormal } 2(x) \Rightarrow \neg \text{Pacifist}(x) . \\ & \text{Quaker}(x) \wedge \neg \text{Abnormal } 3(x) \Rightarrow \text{Pacifist}(x) . \end{aligned}$$

If we circumscribe Abnormal 2 and Abnormal 3, there are two preferred models: one in which Abnormal 2(Nixon) and Pacifist(Nixon) hold and one in which Abnormal 3(Nixon) and $\neg \text{Pacifist}(\text{Nixon})$ hold. Thus, the circumscriptive reasoner remains properly agnostic as to whether Nixon was a pacifist. If we wish, in addition, to assert that religious beliefs take precedence over political beliefs, we can use a formalism called prioritized circumscription to give preference to models where Abnormal 3 is minimized.

Default logic is a formalism in which default rules can be written to generate contingent, nonmonotonic conclusions. A default rule looks like this:

$$\text{Bird}(x) : \text{Flies}(x) / \text{Flies}(x) .$$

This rule means that if Bird(x) is true, and if Flies(x) is consistent with the knowledge base, then Flies(x) may be concluded by default. In general, a default rule has the form

$$P : J_1, \dots, J_n / C$$

where P is called the prerequisite, C is the conclusion, and J_i is the justification—if any one of them can be proven false, then the conclusion cannot be drawn. Any variable that appears in J_i or C must also appear in P. The Nixon-diamond example can be represented in default logic with one fact and two default rules:

$$\begin{aligned} & \text{Republican}(\text{Nixon}) \wedge \text{Quaker}(\text{Nixon}) . \\ & \text{Republican}(x) : \neg \text{Pacifist}(x) / \neg \text{Pacifist}(x) . \\ & \text{Quaker}(x) : \text{Pacifist}(x) / \text{Pacifist}(x) . \end{aligned}$$

To interpret what the default rules mean, we define the notion of an extension of a default theory to be a maximal set of consequences of the theory. That is, an extension S consists of the original known facts and a set of conclusions from the default rules, such that no additional conclusions can be drawn from S and the justifications of every default conclusion in S are consistent with S. As in the case of the preferred models in circumscription, we have two possible extensions for the Nixon diamond: one wherein he is a pacifist and one wherein he is not. Prioritized schemes exist in which some default rules can be given precedence over others, allowing some ambiguities to be resolved.

Since 1980, when nonmonotonic logics were first proposed, a great deal of progress has been made in understanding their mathematical properties. There are still unresolved questions, however. For example, if “Cars have four wheels” is false, what does it mean to have it in one’s knowledge base? What is a good set of

default rules to have? If we cannot decide, for each rule separately, whether it belongs in our knowledge base, then we have a serious problem of non modularity. Finally, how can beliefs that have default status be used to make decisions? This is probably the hardest issue for default reasoning. Decisions often involve trade offs, and one therefore needs to compare the strengths of belief in the outcomes of different actions, and the costs of making a wrong decision. In cases where the same kinds of decisions are being made repeatedly, it is possible to interpret default rules as “threshold probability” statements. For example, the default rule “My brakes are always OK” really means “The probability that my brakes are OK, given no other information, is sufficiently high that the optimal decision is for me to drive without checking them.” When the decision context changes—for example, when one is driving a heavily laden truck down a steep mountain road—the default rule suddenly becomes inappropriate, even though there is no new evidence of faulty brakes. These considerations have led some researchers to consider how to embed default reasoning within probability theory or utility theory.

Truth maintenance systems

We have seen that many of the inferences drawn by a knowledge representation system will have only default status, rather than being absolutely certain. Inevitably, some of these inferred facts will turn out to be wrong and will have to be retracted in the face of new information. This process is called belief revision. Suppose that a knowledge base KB contains a sentence P—perhaps a default conclusion recorded by a forward-chaining algorithm, or perhaps just an incorrect assertion—and we want to execute $\text{TELL}(\text{KB}, \neg P)$. To avoid creating a contradiction, we must first execute $\text{RETRACT}(\text{KB}, P)$. This sounds easy enough. Problems arise, however, if any additional sentences were inferred from P and asserted in the KB. For example, the implication $P \Rightarrow Q$ might have been used to add Q. The obvious “solution”—retracting all sentences inferred from P—fails because such sentences may have other justifications besides P. For example, if R and $R \Rightarrow Q$ are also in the KB, then Q does not have to be removed after all. Truth maintenance systems, or TMSs, are designed to handle exactly these kinds of complications.

One simple approach to truth maintenance is to keep track of the order in which sentences are told to the knowledge base by numbering them from P_1 to P_n . When the call $\text{RETRACT}(\text{KB}, P_i)$ is made, the system reverts to the state just before P_i was added, thereby removing both P_i and any inferences that were derived from P_i . The sentences P_{i+1} through P_n can then be added again. This is simple, and it guarantees that the knowledge base will be consistent, but retracting P_i requires retracting and reasserting $n - i$ sentences as well as undoing and redoing all the inferences drawn from those sentences. For systems to which many facts are being added—such as large commercial databases—this is impractical.

A more efficient approach is the justification-based truth maintenance system, or JTMS. In a JTMS, each sentence in the knowledge base is annotated with a justification consisting of the set of sentences from which it was inferred. For example, if the knowledge base already contains $P \Rightarrow Q$, then $\text{TELL}(P)$ will cause Q to be added with the justification $\{P, P \Rightarrow Q\}$. In general, a sentence can have any number of justifications. Justifications make retraction efficient. Given the call $\text{RETRACT}(P)$, the JTMS will delete exactly those sentences for which P is a member of every justification. So, if a sentence Q had the single justification $\{P, P \Rightarrow Q\}$, it would be removed; if it had the additional justification $\{P, P \vee R \Rightarrow Q\}$, it would still be removed; but if it also had the justification $\{R, P \vee R \Rightarrow Q\}$, then it would be spared. In this way, the time required for retraction of P depends only on the number of sentences derived from P rather than on the number of other sentences added since P entered the knowledge base.

The JTMS assumes that sentences that are considered once will probably be considered again, so rather than deleting a sentence from the knowledge base entirely when it loses all justifications, we merely mark the sentence as being out of the knowledge base. If a subsequent assertion restores one of the justifications, then we mark the sentence as being back in. In this way, the JTMS retains all the inference chains that it uses and need not rederive sentences when a justification becomes valid again.

In addition to handling the retraction of incorrect information, TMSs can be used to speed up the analysis of multiple hypothetical situations. Suppose, for example, that the Romanian Olympic Committee is choosing sites for the swimming, athletics, and equestrian events at the 2048 Games to be held in Romania. For example, let the first hypothesis be Site(Swimming,Pitesti), Site(Athletics, Bucharest), and Site(Equestrian, Arad). A great deal of reasoning must then be done to work out the logistical consequences and hence the desirability of this selection. If we want to consider Site(Athletics, Sibiu) instead, the TMS avoids the need to start again from scratch. Instead, we simply retract Site(Athletics, Bucharest) and assert Site(Athletics, Sibiu) and the TMS takes care of the necessary revisions. Inference chains generated from the choice of Bucharest can be reused with Sibiu, provided that the conclusions are the same.

An assumption-based truth maintenance system, or ATMS, makes this type of context switching between hypothetical worlds particularly efficient. In a JTMS, the maintenance of justifications allows you to move quickly from one state to another by making a few retractions and assertions, but at any time only one state is represented. An ATMS represents all the states that have ever been considered at the same time. Whereas a JTMS simply labels each sentence as being in or out, an ATMS keeps track, for each sentence, of which assumptions would cause the sentence to be true. In other words, each sentence has a label that consists of a set of assumption sets. The sentence holds just in those cases in which all the assumptions in one of the assumption sets hold.

Truth maintenance systems also provide a mechanism for generating explanations. Technically, an explanation of a sentence P is a set of sentences E such that E entails P. If the sentences in E are already known to be true, then E simply provides a sufficient basis for proving that P must be the case. But explanations can also include assumptions of sentences that are not known to be true, but would suffice to prove P if they were true. For example, one might not have enough information to prove that one's car won't start, but a reasonable explanation might include the assumption that the battery is dead. This, combined with knowledge of how cars operate, explains the observed non behavior. In most cases, we will prefer an explanation E that is minimal, meaning that there is no proper subset of E that is also an explanation. An ATMS can generate explanations for the "car won't start" problem by making assumptions (such as "gas in car" or "battery dead") in any order we like, even if some assumptions are contradictory. Then we look at the label for the sentence "car won't start" to read off the sets of assumptions that would justify the sentence.

The exact algorithms used to implement truth maintenance systems are a little complicated, and we do not cover them here. The computational complexity of the truth maintenance problem is at least as great as that of propositional inference—that is, NP-hard. Therefore, you should not expect truth maintenance to be a panacea. When used carefully, however, a TMS can provide a substantial increase in the ability of a logical system to handle complex environments and hypotheses.