UNIT-III

(1) How Knowledge is represented?

A variety of ways of knowledge(facts) have been exploited in AI programs.Facts : truths in some relevant world. These are things we want to represent.

(2) What is propositional logic?

It is a way of representing knowledge. In logic and mathematics, a **propositional calculus** or **logic** is a formal system in which formulae representing *propositions* can be formed by combining atomic propositions using *logical connectives* Sentences considered in propositional logic are not arbitrary sentences but are the ones that are either true or false, but not both. This kind of sentences are called **propositions**.

Example

Some facts in propositional logic:

It is raning. - RAINING

It is sunny - SUNNY

It is windy - WINDY

If it is raining ,then it is not sunny - RAINING -> ¬ SUNNY

Elements of propositional logic

Simple sentences which are true or false are basic propositions. Larger and more complex sentences are constructed from basic propositions by combining them with **connectives**. Thus **propositions** and **connectives** are the basic elements of propositional logic. Though there are many connectives, we are going to use the following **five basic connectives** here:

NOT, AND, OR, IF_THEN (or IMPLY), IF_AND_ONLY_IF.

They are also denoted by the symbols:

¬, ∧, ∨, →, ↔, respectively.

Inference is deriving new sentences from old.

Modus ponens

There are standard patterns of inference that can be applied to derive chains of conclusions that lead to the desired goal. These patterns of inference are called **inference rules.** The best-known rule is called **Modus Ponens** and is written as follows:

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$

The notation means that, whenever any sentences of the form $\alpha \Rightarrow \beta$ and α are given, then the sentence β can be inferred. For example, if $(WumpusAhead \wedge WumpusAlive) \Rightarrow Shoot$ and $(WumpusAhead \wedge WumpusAlive)$ are given, then $Shoot$ can be inferred.

Another useful inference rule is **And-Elimination**, which says that, from a conjunction, any of the conjuncts can be inferred:

$$\frac{\alpha \wedge \beta}{\alpha}$$

For example, from $(WumpusAhead \wedge WumpusAlive)$, $WumpusAlive$ can be inferred.

Entailment

Propositions tell about the notion of truth and it can be applied to logical reasoning.We can have logical entailment between sentences. This is known as entailment where a sentence follows logically from another sentence. In mathematical notation we write :

$$\alpha \models \beta$$

Knowledge based agents

The central component of a knowledge-based agent is its knowledge base, or KB. Informally, a knowledge base is a set of sentences. Each sentence is expressed in a language called a knowledge representation language and represents some assertion about the world.

```
function KB-AGENT(percept) returns an action
  static: KB, a knowledge base
          t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION- QUERY(^))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

Figure 7.1 A generic knowledge-based agent.

Figure 7.1 shows the outline of a knowledge-based agent program. Like all our agents, it takes a percept as input and returns an action. The agent maintains a knowledge base, KB, which may initially contain some **background knowledge.** Each time the agent program is called, it does three things. First, it TELLS the knowledge base what it perceives. Second, it ASKS the knowledge base what action it should perform. In the process of answering this query, extensive reasoning may be done about the current state of the world, about the outcomes of possible action sequences, and so on.

Connectives used in propositional logic.

The **syntax** of propositional logic defines the allowable sentences. The **atomic sentences-** the indivisible syntactic elements-consist of a single **proposition symbol.** Each such symbol stands for a proposition that can be true or false. We will use uppercase names for symbols: P, Q, R, and so on.

Complex sentences are constructed from simpler sentences using **logical connectives.**

There are five connectives in common use:

¬ (not). A sentence such as $\neg W_{1,3}$ is called the **negation** of $W_{1,3}$. **A literal** is either an atomic sentence (a **positive literal)** or a negated atomic sentence (a **negative literal).**

∧ (and). A sentence whose main connective is ∧, such as $W_{1,3} \wedge P_{3,1}$, is called a **conjunction;** its parts are the **conjuncts.** (The ∧ looks like an "A" for "And.")

∨ (or). A sentence using ∨, such as $(W_{1,3} \wedge P_{3,1}) \vee W_{2,2}$, is a **disjunction** of the **disjuncts** $(W_{1,3} \wedge P_{3,1})$ and $W_{2,2}$. (Historically, the ∨ comes from the Latin "vel," which means "or." For most people, it is easier to remember as an upside-down ∧.)

⇒ (implies). A sentence such as $(W_{1,3} \wedge P_{3,1}) \Rightarrow \neg W_{2,2}$ is called an **implication** (or conditional). Its **premise** or **antecedent** is $(W_{1,3} \wedge P_{3,1})$, and its **conclusion** or **consequent** is $\neg W_{2,2}$. Implications are also known as **rules** or **if–then** statements. The implication symbol is sometimes written in other books as ⊃ or →.

⇔ (if and only if). The sentence $W_{1,3} \Leftrightarrow \neg W_{2,2}$ is a **biconditional.**

Figure 7.7 gives a formal grammar of propositional logic;

```
Sentence        → AtomicSentence | ComplexSentence
AtomicSentence  → True | False | Symbol
Symbol          → P | Q | R | ...
ComplexSentence → ¬ Sentence
                | ( Sentence ∧ Sentence )
                | ( Sentence ∨ Sentence )
                | ( Sentence ⇒ Sentence )
                | ( Sentence ⇔ Sentence )
```

Figure 7.7 A BNF (Backus–Naur Form) grammar of sentences in propositional logic.

First order Logic

Whereas propositional logic assumes the world contains facts, first-order logic (like natural language) assumes the world contains Objects: people, houses, numbers, colors, baseball games, wars, … Relations: red, round, prime, brother of, bigger than, part of, comes between, … Functions: father of, best friend, one more than, plus, …

Specify the syntax of First-order logic in BNF form.

$$
\begin{aligned}
Sentence &\rightarrow AtomicSentence \\
&\mid (\ Sentence\ Connective\ Sentence\ ) \\
&\mid Quantifier\ Variable,\ldots\ Sentence \\
&\mid \neg\ Sentence \\
\\
AtomicSentence &\rightarrow Predicate(Term,\ldots) \mid Term = Term \\
\\
Term &\rightarrow Function(Term,\ldots) \\
&\mid Constant \\
&\mid Variable \\
\\
Connective &\rightarrow \Rightarrow \mid \wedge \mid \vee \mid \Leftrightarrow \\
Quantifier &\rightarrow \forall \mid \exists \\
Constant &\rightarrow A \mid X_1 \mid John \mid \ldots \\
Variable &\rightarrow a \mid x \mid s \mid \ldots \\
Predicate &\rightarrow Before \mid HasColor \mid Raining \mid \ldots \\
Function &\rightarrow Mother \mid LeftLeg \mid \ldots
\end{aligned}
$$

Figure 8.3 The syntax of first-order logic with equality, specified in Backus–Naur form. (See page 984 if you are not familiar with this notation.) The syntax is strict about parentheses; the comments about parentheses and operator precedence on page 205 apply equally to first-order logic.

Compare different knowledge representation languages.

Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes about facts)
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $\in [0, 1]$
Fuzzy logic	facts with degree of truth $\in [0, 1]$	known interval value

Figure 8.1 Formal languages and their ontological and epistemological commitments.

Syntactic elements of First Order Logic

The basic syntactic elements of first-order logic are the symbols that stand for objects,relations, and functions. The symbols,come in three kinds:

a) constant symbols, which stand for objects;

b) predicate symbols, which stand for relations;

c) and function symbols, which stand for functions.

We adopt the convention that these symbols will begin with uppercase letters.

Example:

Constant symbols :

Richard and John;

predicate symbols :

Brother, OnHead, Person, King, and Crown;

function symbol :

LeftLeg.

Quantifiers:-

There is need to express properties of entire collections of objects,instead of enumerating the objects by name. Quantifiers let us do this.

FOL contains two standard quantifiers called

a) Universal (∀**)** and

b) Existential (∃**)**

Universal quantification

(∀x) P(x) : means that P holds for **all** values of x in the domain associated with that variable

E.g., (∀x) dolphin(x) => mammal(x)

Existential quantification

(∃ x)P(x) means that P holds for **some** value of x in the domain associated with that variable

E.g., (∃ x) mammal(x) ^ lays-eggs(x)

Permits one to make a statement about some object without naming it

UNIVERSAL QUANTIFIERS WITH AN EXAMPLE.

Rules such as "All kings are persons," is written in first-order logic as

∀x King(x) => Person(x)

where ∀ is pronounced as " For all .."

Thus, the sentence says, "For all *x,* if *x* is a king, then z is a person."

The symbol x is called a variable(lower case letters)

The sentence ∀**x** P,where P is a logical expression says that P is true for every object x.

Existential quantifiers with an example.

Universal quantification makes statements about every object. It is possible to make a statement about some object in the universe without naming it,by using an existential quantifier.

Example

"King John has a crown on his head"

∃x Crown(x) ^ OnHead(x,John)

∃x is pronounced "There exists an x such that .." or " For some x .."

NESTED QUANTIFIERS:-

Nested quantifiers

We will often want to express more complex sentences using multiple quantifiers. The simplest case is where the quantifiers are of the same type. For example, "Brothers are siblings" can be written as

$$\forall x \ \forall y \ \ Brother(x, y) \Rightarrow Sibling(x, y).$$

Example-2

"Everybody loves somebody" means that

for every person,there is someone that person loves

$\forall x \exists y$ Loves(x,y)

Connection between $\forall$ and $\exists$

"Everyone likes icecream " is equivalent

"there is no one who does not like ice cream"

This can be expressed as :

$\forall x$ Likes(x,IceCream) is equivalent to

$\neg \exists \neg$Likes(x,IceCream)

Knowledge Engineering process:-

Discuss them by applying the steps to any real world application of your choice.

Knowledge Engineering

The general process of knowledge base constructiona process is called knowledge engineering.

A knowledge engineer is someone who investigates a particular domain, learns what concepts are important in that domain, and creates a formal representation of the objects and relations in the domain. We will illustrate the knowledge

engineering process in an electronic circuit domain that should already be fairly familiar,

The steps associated with the knowledge engineering process are :

1. ***Identfy the task***.

. The task will determine what knowledge must be represented in order to connect problem instances to answers. This step is analogous to the PEAS process for designing agents.

2. ***Assemble the relevant knowledge***. The knowledge engineer might already be an expert in the domain, ***or*** might need to work with real experts to extract what they know-a process called **knowledge acquisition.**

3. ***Decide on a vocabulary of predicates, functions, and constants***. That is, translate the important domain-level concepts into logic-level names.

Once the choices have been made. the result is a vocabulary that is known as the **ontology** of

the domain. The word *ontology* means a particular theory of the nature of being or existence.

4. Encode general /cnowledge about the domain. The knowledge engineer writes down the axioms for all the vocabulary terms. This pins down (to the extent possible) the meaning of the terms, enabling the expert to check the content. Often, this step reveals misconceptions or gaps in the vocabulary that must be fixed by returning to step 3 and iterating through the process.

5. Encode a description of the specijic problem insttznce.

For a logical agent, problem instances are supplied by the sensors, whereas a "disembodied" knowledge base is supplied with additional sentences in the same way that traditional programs are supplied with input data.

6. *Pose queries to the inference procedure and get answers.* This is where the reward is: we can let the inference procedure operate on the axioms and problem-specific facts to derive the facts we are interested in knowing.

7. Debug the knowledge base.

$\forall x$ *NumOfLegs(x,4) => Mammal(x)*

Is false for reptiles ,amphibians.

To understand this seven-step process better, we now apply it to an extended example-the domain of electronic circuits.

The electronic circuits domain

We will develop an ontology and knowledge base that allow us to reason about digital circuits of the kind shown in Figure 8.4. We follow the seven-step process for knowledge engineering.

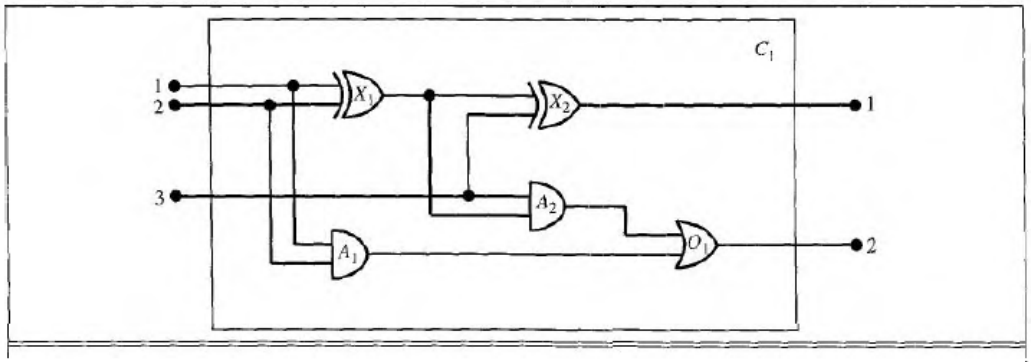


Figure 8.4 A digital circuit C1, purporting to be a one-bit full adder. The first two inputs are the two bits to be added and the third input is a carry bit. The first output is the sum, and the second output is a carry bit for the next adder. The circuit contains two XOR gates, two AND gates and one OR gate.

Identify the task

There are many reasoning tasks associated with digital circuits. At the highest level, one analyzes the circuit's functionality. For example, what are all the gates connected to the first input terminal? Does the circuit contain feedback loops? These will be our tasks in this section.

Assemble the relevant knowledge

What do we know about digital circuits? For our purposes, they are composed of wires and gates. Signals flow along wires to the input terminalls of gates, and each gate produces a signal on the output terminal that flows along another wire.

Decide on a vocabulary

We now know that we want to talk about circuits, terminals, signals, and gates. The next step is to choose functions, predicates, and constants to represent them. We will start from individual gates and move up to circuits. First, we need to be able to distinguish a gate from other gates. This is handled by naming gates with constants: *X I , X2,* and so on

Encode general knowledge of the domain

One sign that we have a good ontology is that there are very few general rules which need to be specified. A sign that we have a good vocabulary is that each rule can be stated clearly and concisely. With our example, we need only seven simple rules to describe everything we need to know about circuits:

1. If two terminals are connected, then they have the same signal:
2. The signal at every terminal is either 1 or 0 (but not both):
3. Connected is a commutative predicate:
4. An OR gate's output is 1 if and only if any of its inputs is 1:
5. An **A.ND** gate's output is 0 if and only if any of its inputs is 0:
6. An XOR gate's output is 1 if and only if its inputs are different:
7. A NOT gate's output is different from its input:

Encode the specific problem instance

The circuit shown in Figure 8.4 is encoded as circuit *C1* with the following description. First, we categorize the gates: *Type(X1)= XOR Type(X2)= XOR*

Pose queries to the inference procedure

What combinations of inputs would cause the first output of *Cl* (the sum bit) to be 0 and the second output of *C1* (the carry bit) to be l?

Debug the knowledge base

We can perturb the knowledge base in various ways to see what kinds of erroneous behaviors emerge.

Usage of First Order Logic.

The best way to find usage of First order logic is through examples. The examples can be taken from some simple **domains.** In knowledge representation, a domain is just some part of the world about which we wish to express some knowledge.

Assertions and queries in first-order logic

Sentences are added to a knowledge base using TELL, exactly as in propositional logic. Such sentences are called **assertions.**

For example, we can assert that John is a king and that kings are persons:

TELL(KB, King (John)) .

Where KB is knowledge base.

TELL(KB, ∀x King(x) => Person(x)).

We can ask questions of the knowledge base using ASK. For example,

$$\text{ASK}(KB,\ King(John))$$

returns *true.* Questions asked using ASK are called **queries** or **goals**

ASK(KB,Person(John))

Will return true.

(ASK KBto find whther Jon is a king)

ASK(KB, ∃**x** person(x))

The kinship domain

The first example we consider is the domain of family relationships, or kinship. This domain includes facts such as "Elizabeth is the mother of Charles" and "Charles is the father of William7' and rules such as "One's grandmother is the mother of one's parent."

Clearly, the objects in our domain are people. We will have two unary predicates, *Male* and *Female.* Kinship relations-parenthood, brotherhood, marriage, and so on-will be represented by binary predicates: *Parent, Sibling, Brother, Sister, Child, Daughter,Son, Spouse, Husband, Grandparent, Grandchild, Cousin, Aunt,* and *Uncle.*

We will use functions for *Mother* and *Father.*

UNIVERSAL INSTANTIATION:-

Inference rules for quantifiers

Let us begin with universal quantifiers. Suppose our knowledge base contains the standard folkloric axiom stating that all greedy kings are evil:

$$\forall x \;\; King(x) \wedge Greedy(x) \Rightarrow Evil(x) .$$

Then it seems quite permissible to infer any of the following sentences:

$$King(John) \wedge Greedy(John) \Rightarrow Evil(John).$$
$$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard) .$$
$$King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John)).$$

The rule of **Universal Instantiation** (UI for short) says that we can infer any sentence obtained by substituting a **ground term** (a term without variables) for the variable. [1] To write out the inference rule formally, we use the notion of **substitutions** introduced in Section 8.3. Let $\text{SUBST}(\theta, a)$ denote the result of applying the substitution θ to the sentence a. Then the rule is written

$$\frac{\forall v \;\; a}{\text{SUBST}(\{v/g\}, \alpha)}$$

for any variable v and ground term g. For example, the three sentences given earlier are obtained with the substitutions $\{x/John\}$, $\{x/Richard\}$, and $\{x/Father(John)\}$.

The corresponding **Existential Instantiation** rule: for the existential quantifier is slightly more complicated. For any sentence α, variable v, and constant symbol k that does not appear elsewhere in the knowledge base,

$$\frac{\exists v \;\; \alpha}{\text{SUBST}(\{v/k\}, \alpha)} .$$

Universal instantiation (UI)

- Every instantiation of a universally quantified sentence is entailed by it:

$$\frac{\forall v\ \alpha}{\text{Subst}(\{v/g\}, \alpha)}$$

  for any variable *v* and ground term *g*

- E.g., $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ yields:

  $King(John) \wedge Greedy(John) \Rightarrow Evil(John)$
  $King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$
  $King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John))$
  .
  .
  .

Existential instantiation (EI)

- For any sentence α, variable *v*, and constant symbol *k* that does not appear elsewhere in the knowledge base:

$$\frac{\exists v\ \alpha}{\text{Subst}(\{v/k\}, \alpha)}$$

- E.g., $\exists x\ Crown(x) \wedge OnHead(x,John)$ yields:

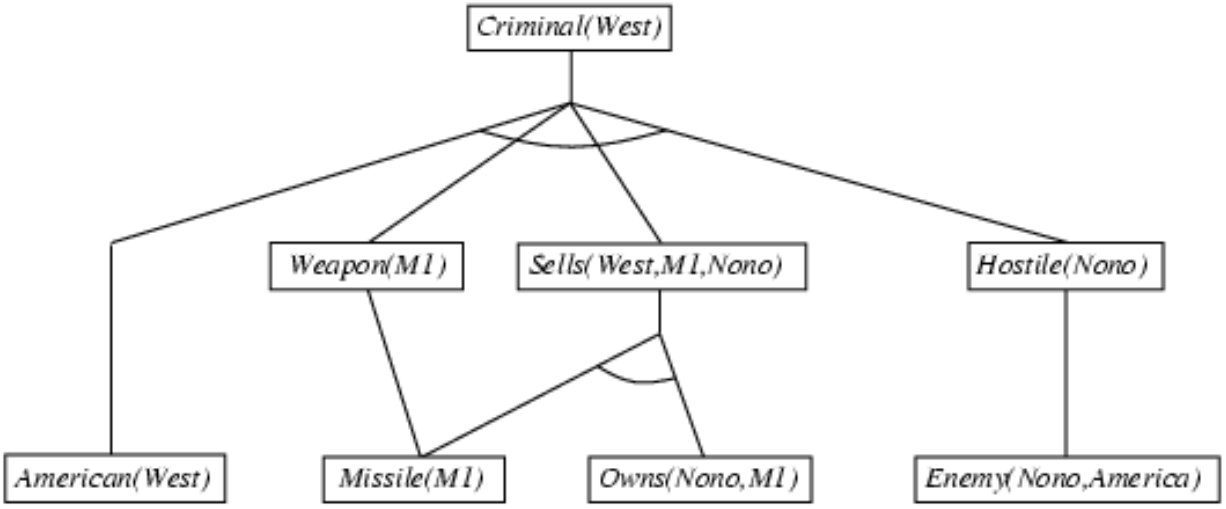
$$Crown(C_1) \wedge OnHead(C_1,John)$$

provided C_1 is a new constant symbol, called a Skolem constant

FORWARD CHAININ:-

Using a deduction to reach a conclusion from a set of antecedents is called forward chaining. In other words,the system starts from a set of facts,and a set of rules,and tries to find the way of using these rules and facts to deduce a conclusion or come up with a suitable couse of action. This is known as data driven reasoning.

EXAMPLE



The proof tree generated by forward chaining.

Example knowledge base

- The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

- Prove that Col. West is a criminal

... it is a crime for an American to sell weapons to hostile nations:
American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)
Nono … has some missiles, i.e., ∃x Owns(Nono,x) ∧ Missile(x):
Owns(Nono,M_1) and Missile(M_1)
… all of its missiles were sold to it by Colonel West
Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)
Missiles are weapons:
Missile(x) ⇒ Weapon(x)
An enemy of America counts as "hostile“:
Enemy(x,America) ⇒ Hostile(x)
West, who is American …
American(West)
The country Nono, an enemy of America …
Enemy(Nono,America)

Note:

(a) The initial facts appear in the bottom level

(b) Facts inferred on the first iteration is in the middle level

(c) The facts inferered on the 2nd iteration is at the top levelForward chaining algorithm

BACKWARD CHAINING :-

Forward chaining applies a set of rules and facts to deduce whatever conclusions can be derived.

In **backward chaining** ,we start from a **conclusion**,which is the hypothesis we wish to prove,and we aim to show how that conclusion can be reached from the rules and facts in the data base.

The conclusion we are aiming to prove is called a goal ,and the reasoning in this way is known as **goal-driven**.

Backward chaining example

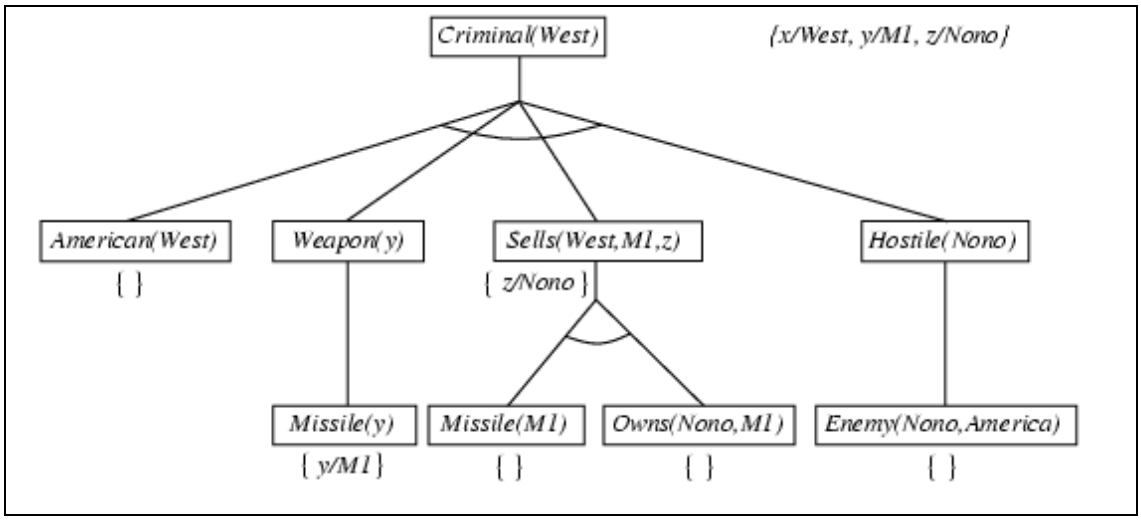


Fig : Proof tree constructed by backward chaining to prove that West is criminal.

Note:

(a) To prove Criminal(West) ,we have to prove four conjuncts below it.

(b) Some of which are in knowledge base,and others require further backward chaining.

(3) Explain conjunctive normal form for first-order logic with an example.

Conjunctive normal form for first-order logic

As in the propositional case, first-order resolution requires that sentences be in **conjunctive normal form** (CNF)—that is, a conjunction of clauses, where each clause is a disjunction of literals.[6] Literals can contain variables, which are assumed to be universally quantified. For example, the sentence

$$\forall x \;\; American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

becomes, in CNF,

$$\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x, y, z) \vee \neg Hostile(z) \vee Criminal(x).$$

Every sentence of first-order logic can be converted into an inferentially equivalent CNF sentence. In particular, the CNF sentence will be unsatisfiable just when the original sentence is unsatisfiable, so we have a basis for doing proofs by contradiction on the CNF sentences.

Here we have to eliminate existential quantifiers. We will illustrate the procedure by translating the sentence "Everyone who loves all animals is loved by someone," or

$$\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$$

The steps are as follows:

- Eliminate implications:

$$ti\,x\ [\neg\forall y\ \neg Animal(y) \vee Loves(x,y)]\ \mathbf{v}\ [\exists y\ Loves(y,x)]$$

- Move $\neg$ inwards: In addition to the usual rules for negated connectives, we need rules for negated quantifiers. Thus, we have

$\neg\forall x\ p$ becomes $3x\ \neg p$
$\neg\exists x\ p$ becomes $\forall x\ \neg p$.

Our sentence goes through the following transformations:

$$\forall x\ [\exists y\ \neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y\ Loves(y,x)].$$
$$\forall x\ [\exists y\ \neg\neg Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\ Loves(y,x)].$$
$$\forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x,y)]\ \mathbf{v}\ [\exists y\ Loves(y,x)].$$

Notice how a universal quantifier $(\forall y)$ in the premise of the implication has become an existential quantifier. The sentence now reads "Either there is some animal that x doesn't love, or (if this is not the case) someone loves x." Clearly, the meaning of the original sentence has been preserved.

ONTOLOGICAL ENGINEERING:-

Ontology refers to organizing every thing in the world into hierarch of categories. Representing the abastract concepts such as Actions,Time,Physical Objects,and Beliefs is called Ontological Engineering.

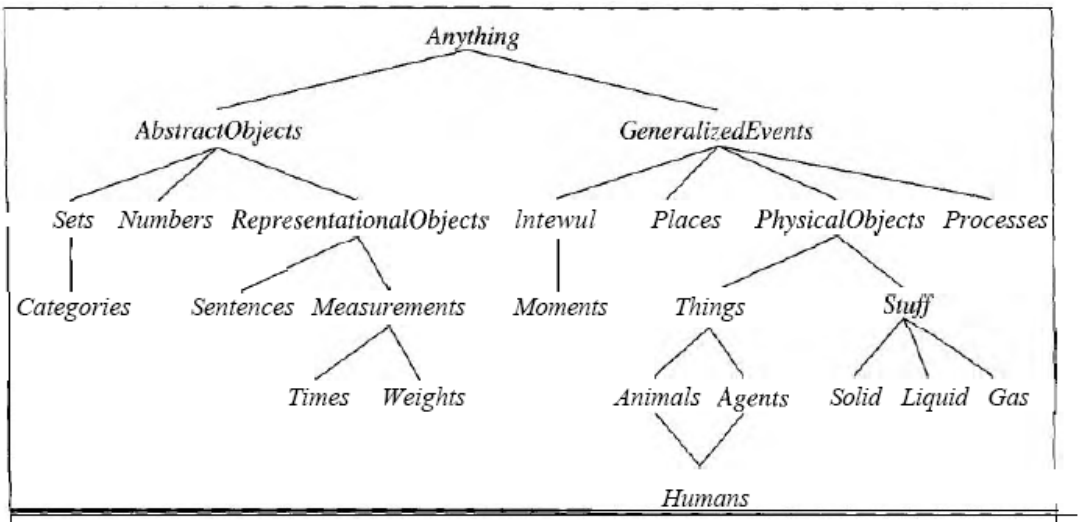


Figure 10.1 The upper ontology of the world, showing the topics to be covered later in the chapter. Each arc indicates that the lower concept is a specialization of the upper one.

categories are useful in Knowledge representation

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.

TAXONOMY:-

Subclass relations organize categories into a taxonomy, or taxonomic hierarchy. Taxonomies have been used explicitly for centuries in technical fields. For example, systematic biology aims to provide a taxonomy of all living and extinct species; library science has developed a taxonomy of all fields of knowledge, encoded as the Dewey Decimal system; and tax authorities and other government departments have developed extensive taxoriornies of occupations and commercial products. Taxonomies are also an important aspect of general commonsense knowledge.

First-order logic makes it easy to state facts about categories, either by relating objects to categories or by quantifying over their members:

- An object is a member of a category. For example:
  $BB_9 \in Basketballs$
- A category is a subclass of another category. For example:
  $Basketballs \subset Balls$
- All members of a category have some properties. For example:
  $x \in Basketballs \Rightarrow Round(x)$
- Members of a category can be recognized by some properties. For example:
  $Orange(x) \wedge Round(z) \wedge Diameter(x) = 9.5'' \wedge x \in Balls \Rightarrow x \in Basketballs$
- A category as a whole has some properties. For example:
  $Dogs \in DomesticatedSpecies$

:Notice that because *Dogs* is a category and is a member of *DomesticatedSpecies*, the latter must be a category of categories. One can even have categories of categories of categories, but they are not much use.

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:

$PartOf(Bucharest, Romania)$
$PartOf(Romania, EasternEurope)$
$PartOf(EasternEurope, Europe)$
$PartOf(Europe, Earth)$.

The *PartOf* relation is transitive and reflexive; that is,

$PartOf(x,y) \wedge PartOf(y,z) \Rightarrow PartOf(x,z)$.
$PartOf(x,x)$.

Therefore, we can conclude $PartOf(Bucharest, Earth)$.

ONTOLOGY OF SITUATION CALCULUS.:-

Situations are logical terms consisting of the initial situation (usually called *So)* and all situations that are generated by applying an action to a situation. The function *Result(a, s)* (sometimes called *Do)* names the situation that results when action *a* is executed in situation *s*. Figure 10.2 illustrates this idea.

Fluents are functions and predicates that vary from one situation to the next, such as the location of the agent or the aliveness of the wumpus. The dictionary says a fluent is something that fllows, like a liquid. In this use, it means flowing or changing across situations. By convention, the situation is always the last argument of a fluent.

For example, *lHoldzng(G1, So)* says that the agent is not holding the gold GI in the initial situation *So. Age(Wumpus, So)* refers to the wumpus's age in *So.*

Atemporal or **eternal** predicates and functions are also allowed. Examples include the predicate Gold (GI) and the function *LeftLeg Of* (*Wumpus).*

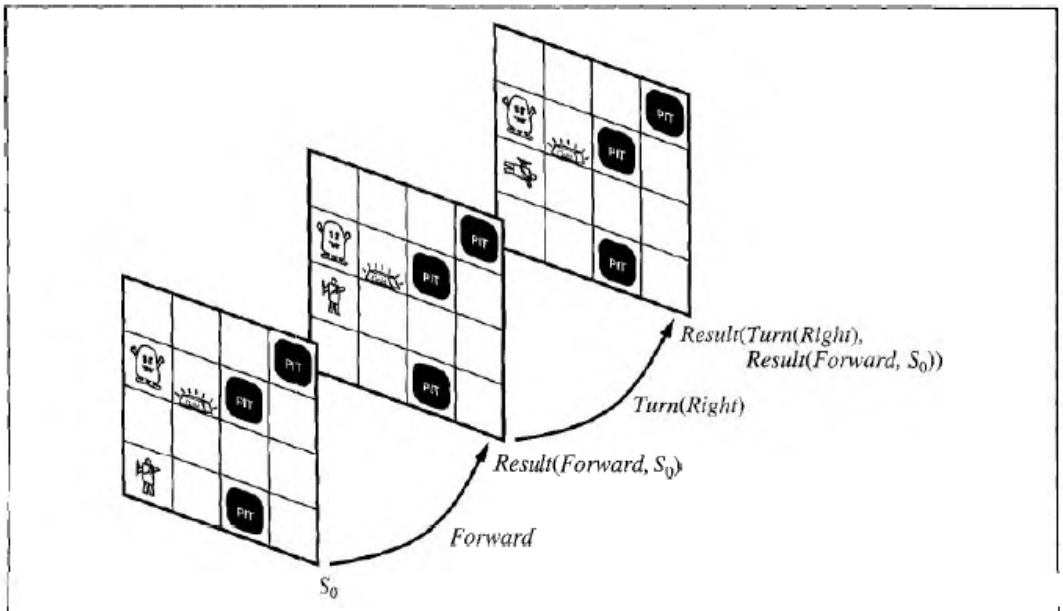


Figure 10.2 In situation calculus, each situation (except S_0) is the result of an action.

Time and event calculus

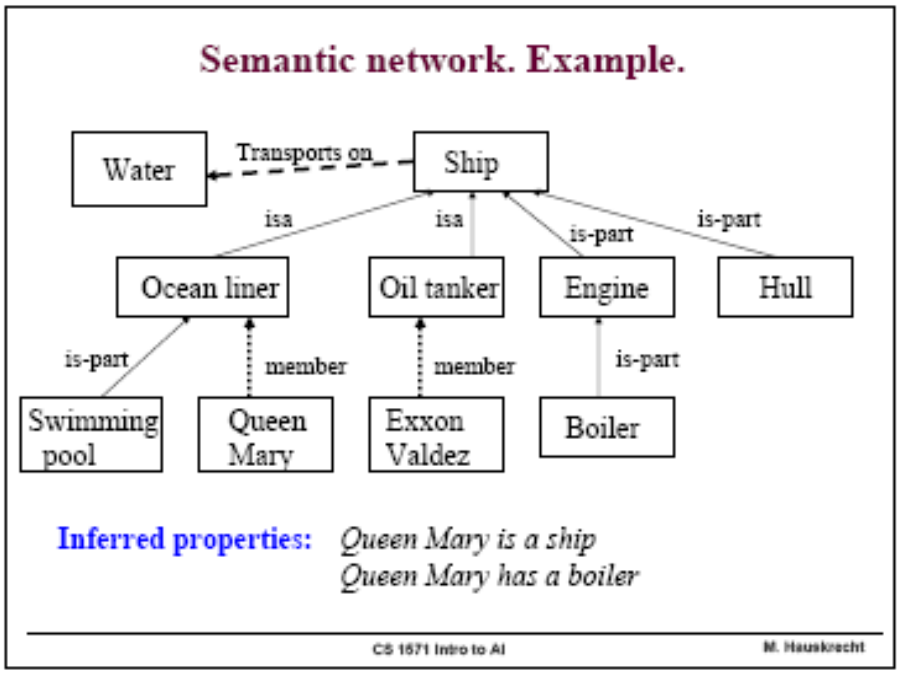
Situation calculus works well when there is a single agent performing instantaneous, discrete actions. When actions have duration and can overlap with each other, situation calculus becomes somewhat awkward. Therefore, we will cover those topics with an alternative for- EVENTCALCULUS malism known as **event calculus,** which is based on points in time rather than on situations. (The terms "event7' and "action" may be used interchangeably. Informally, "event" connotes a wider class of actions, including ones with no explicit agent. These are easier to handle in event calculus than in situation calculus.) In event calculus, fluents hold at points in time rather than at situations, and the calculus is designed to allow reasoning over intervals of time. The event calculus axiom says that a fluent is true at a point in time if the fluent was initiated by an event at some time in the past and was not

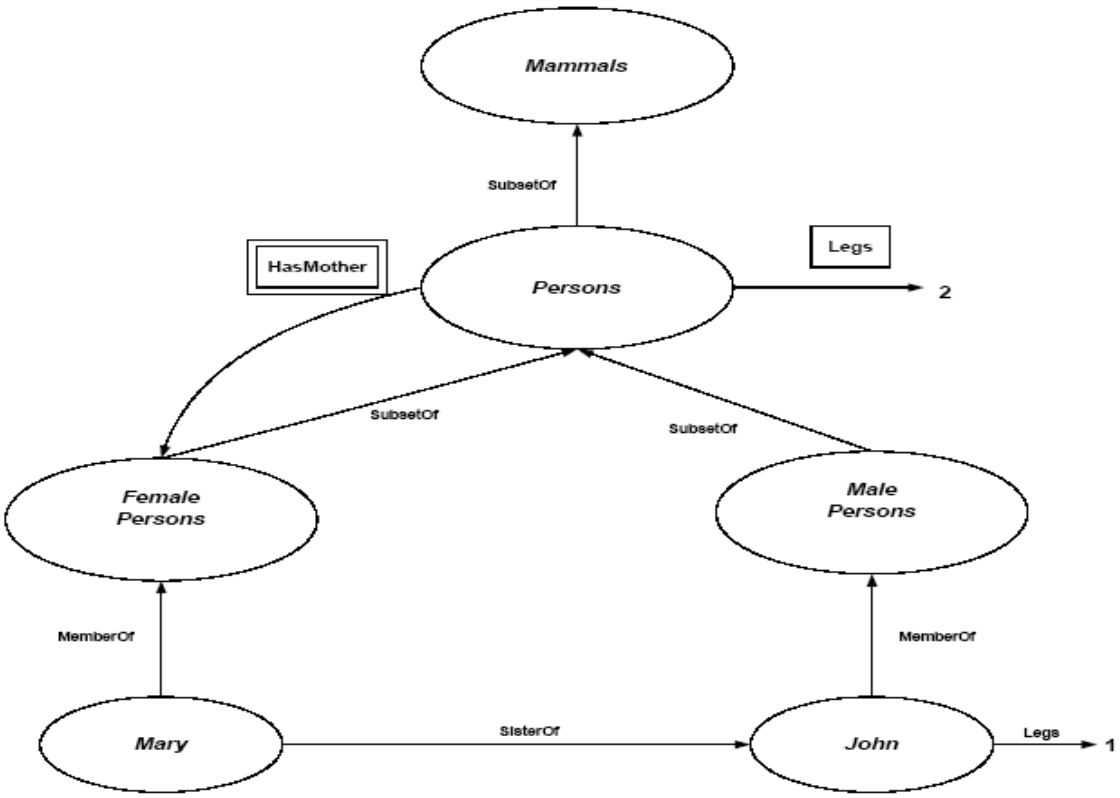
terminated by an intervening event. The *Initiates* and *Terminates* relations play a role similar to the *Result* relation in situation calculus; *Initiates(e, f , t)* means that the occurrence of event *e* at time *t* causes fluent *f* to become true, while *Terminates* (w , f, *t)*means that *f* ceases to be true. We use *Happens(e, t)* to mean that event *e* happens at time *t*,

SEMANTIC NETWORKS:-

Semantic networks are capable of representing individual objects,categories of objects,and relation among objects. Objects or Ctegory names are represented in ovals and are connected by labeled arcs.

Semantic network example




Mammals
SubsetOf
HasMother
Persons
Legs
2
SubsetOf
SubsetOf
Female Persons
Male Persons
MemberOf
MemberOf
Mary
SisterOf
John
Legs
1