CHAPTER

5

Knowledge Representation

Data represents unorganised and unprocessed facts and figures. Usually data is static in nature with a set of discrete facts about events. It simply exists and has no significance beyond its existence. For example, "The price of crude oil is $80 per barrel". From this sentence one may get raw data such as price of crude oil. We cannot say, this is the current price of the crude oil. So, we need an informative phase. Information is processed data. Data that has been interpreted so that it has meaning for the user. Information can be considered as an aggregation of data which makes decision making easier.

For example, "The price of crude oil rises from $80 to $120". This is an information which shows that, at present there is hike in the crude oil by $40.

By the *Oxford dictionary* definition, knowledge is defined as expertise and skills acquired by a person through experience or education, the theoretical or practical understanding of a subject.

Knowledge is human understanding of a subject matter that has been acquired through proper study and experiences. Knowledge is usually built by learning, thinking and proper understanding of the problem area. It can be considered as the integration of human perceptive process that helps them to draw meaningful conclusion. Figure 5.1 shows relation between data, information and knowledge. To explain knowledge, consider the following sentence:

"When crude oil prices go up by $10 per barrel, it is likely that petrol price will rise by 50p per litre", which extends to knowledge.

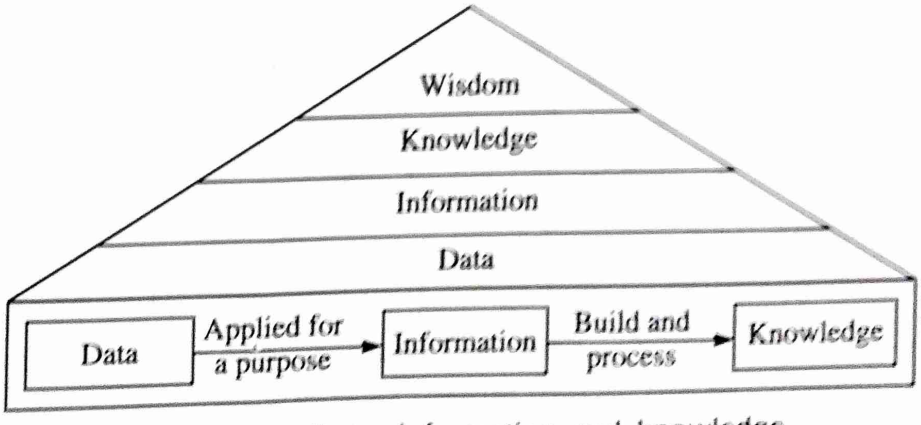

Figure 5.1 Data, information and knowledge.

Wisdom is the ability of human being to judge between right and wrong or good and bad. Wisdom is an understanding and realising of people, things, events or situations, resulting in the capability to choose or find an alternative to constantly create the best results within a minimum time.

The kind of knowledge which needs to be represented in artificial intelligence systems are objects, events, performance, meta-knowledge, facts and knowledgebase. The objects are facts in a domain of real-world (For example, 'a car has wheels'). The actions which occur in our world are events (For example, 'drive a car'). Knowledge about how to do things is understood from performance (For example, 'good mileage for the car'). Meta-knowledge is knowledge about a preselected knowledge or what we know (For example, 'Jimny is an upcoming car from Suzuki'). The truth about real-world representation is called as facts (For example, 'Rolls-Royce is a luxurious car'). Finally, knowledgebase is a group of sentences which gives agent based information. The above example sentences are grouped information of car, which is stored in the knowledgebase. This chapter explains a detailed view of knowledge and its operations.

5.1 Knowledge Management

Knowledge management is a set of strategies and practices used in an organisation to identify, create and represent, distribute and enable adoption of insights and experience. Knowledge management emerged as a scientific discipline in 1990s.

What is knowledge management? It is desirable to address this question after developing some concepts regarding 'knowledge'. Consider the following observation made by Neil Fleming from the DIKW hierarchy in Figure 5.2.

- Collection of data is not information.
- Collection of information is not knowledge.
- Collection of knowledge is not wisdom.
- Collection of wisdom is not truth.

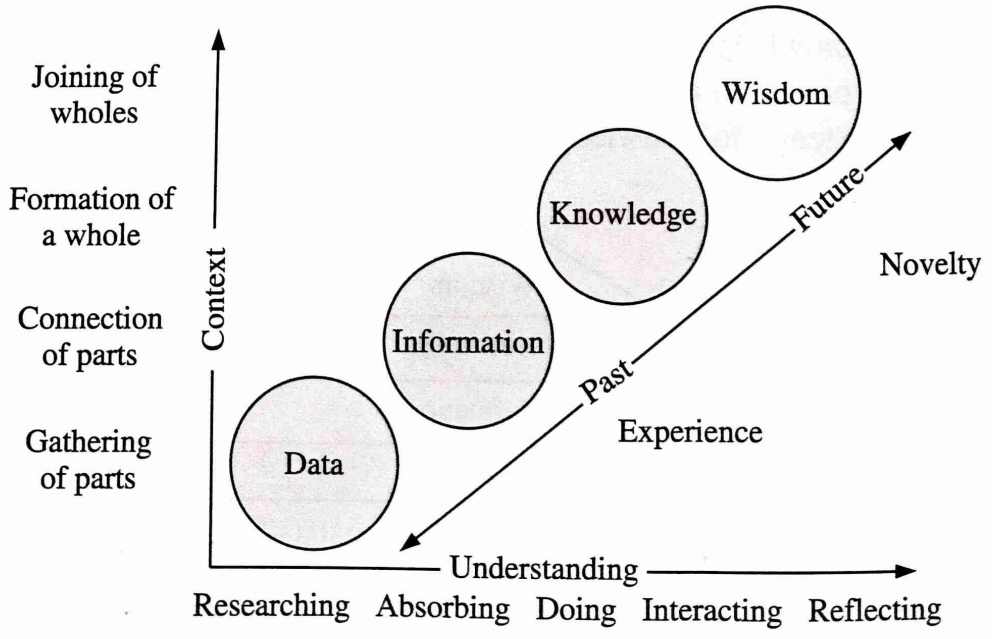

Figure 5.2 DIKW hierarchy.

The basic concept was to view information, knowledge and wisdom not as simple collections but to view as a whole because together it represents more ideas and helps to make more perfect conclusions.

Let us consider an example on stock market and relate the terms data, information, knowledge and wisdom to capital, share value, and profit percentage.

Data: While using numbers like 1000, 2000 or while using 20% or 50% in a situation completely out of context of stock market scenario, these are just pieces of data. The capital, share value, rate and profit percentage can also be termed as data as each of these provides separate meaning according to the context in which they are used.

Information: Data turns to valid information when used in the correct context. The terms capital or share or dividend represent a particular meaning when used in a share market environment. The user can gather more information and draw better conclusion. Here in this situation the following terms are interpreted as:

Capital: means their cash plus the financial assets they have invested.

Investment rate: factor used to compute the extent of investment by a particular company. It can either be a fixed rate or a fixed increase from its previous rate.

Consider a sentence, "Today the share value of Indian oil increased by 1%". This is information for the share holders who invested in the Indian oil company.

Knowledge: Let the capital that an investor possess be rupees 1000. He buys a share for rupees 50. After 2 years the share value increases to 500. Now, the investor sells his share in the market and he gets a profit of rupees 450 from a single share. This now represents a piece of knowledge. By this representation we could easily spot on how the process is evolved and how the output was obtained. By looking at the progressive growth, investor can set the investment rate and can buy the share of companies which has steady and progressive growth. More share the investor possesses larger the chance to get more profit (Considering the fact that the share value is increasing).

With respect to above context, consider the following sentence: "The crude oil usage increases in the Indian market". From this information we have a part of knowledge obtained. "The share value of Indian Oil will increase".

Wisdom: Creating wisdom from knowledge is a big effort. This is the front line principle in developing almost all system principles. The main idea focussed at any action which produces a result that encourages the system to have more of the same action produces. Growth is an everlastingly emergent characteristic for faster or later expansion that runs into limits. Analysis and study of single patterns in the scope will never help in discovering new patterns and creating new learning. Process needs to be a continuous one, which when evolved over time discovers better conclusions and findings.

For example, consider the following sentences:

"The number of vehicles on the Indian roads increases day to day". "In future government subsidies of fuels will be withdrawn".

Knowledge is obtained as "Oil price and share value of Indian Oil Company will increase". From this part of knowledge a wisdom thought can be obtained as "If I purchase more shares of Indian Oil Company, which will be a beneficiary for my future".

5.1.1 Value of Knowledge Management

From an organisational viewpoint, data is anything that represents facts or values of results. Data never form knowledge; the relation between data or the pattern of relationship is the basics of knowledge formation. These relationships need to be represented in a very simple manner so that they are easy to be interpreted. When understood these representations become information or knowledge. This is then updated in the user knowledgebase and stored for future retrieval.

Without associations and related data, it is hard to understand the situations. By associations we are able to perceive the apt meaning. For example, Consider the share of a company ABC, let the initial share value was rupees 500. If we say that the market value of ABC had increased by 50% than the last quarter, then we can easily calculate its present share value. The value could be calculated as we know all the attributes are clearly mentioned. But if we try to answer a question like, whether the share value will shoot up in the next quarter, then we are helpless. This is because although we have data and information, but we lack the knowledge. This is one of the major pitfalls, we are not able to predict the trend of the market. For that, we need more information. We must study the sales trend dealt by the company. Study on the company's immediate competitors, the current products, and the demand of the products is helpful in data analysis. These factors affect the growth of a company and which in-turn reflects in the share value. Various other elements like customer satisfaction, probability of extending the current production area also matters for the same. If we have a full data set like this we can have a data analysis and predictions could be made with most accuracy. Anything less, will create a weak system that has data and information but no knowledge.

From the last example, we can see that past and current data has been used to create or predict the future of share value. This is what knowledge management is about. It basically deals with data analysis of related or associated data for creating a prediction system. They capture, hold and reuse the information and are analysed together to convey meaningful results to a person.

One way of gathering knowledge is through Internet. Knowledge nodes are computers placed in different parts of the world, connected in a large communication network. This knowledge acquisition is done by search engines. Web browsers such as Internet Explorer, Mozilla Firefox, Google Chrome or Opera are major players in this area. The information retrieval is done by crawlers or spiders on the internet which identify and retrieve information from each of the knowledge resources sited in different locations.

5.1.2 Categories of Knowledge

Knowledge can be categorised into the following different ways:

1. Symbolic, sub-symbolic and hybrid

   Symbolic: represented as discrete unit (explicit) expressed in formal languages.

   Sub-symbolic: cannot be represented by single symbols (implicit), stored in a form of information.

   Hybrid: systems involving both types of knowledge representation.

2. Declarative and procedural

Declarative: Knowledge are facts (objects, events) that we observe in the environment. Can describe current state of a problem.

Example: I am a human or the sun shining.

Procedural: knowledge represents the skills or rules we use to operate with and reason using facts.

Example: IF John is a human THEN John has two legs.

5.2 Types of Knowledge

We cannot say there are different types of knowledge. But we are calling two classes of knowledge—declarative knowledge and procedural knowledge. There are three kinds of declarative knowledge—relational knowledge, inheritable knowledge and inferential knowledge. These knowledge is conceptual only and represented in different ways.

5.2.1 Declarative Knowledge

Declarative Knowledge is to know about something with concepts, facts, and objects. The knowledge is represented independently from the mechanisms that will use it as declarative knowledge. This gives a control in adequate use of the knowledge performed by means of general purpose heuristics that determine the best way to use the knowledge. This means the use of knowledge about the control that guides solving mechanism.

Simple relational knowledge

Declarative fact is to represent them as relations which is a simplest way to represent. The same mechanism is used to store in databases too. Databases are intended to provide support for relational knowledge. Even though it gives little opportunity for inference, it may be used as the knowledge basis for inference engines. Simple relational knowledge can be represent in the following ways.

- Store the facts in simple way.
- Set out each fact about a set of objects systematically in columns.
- Little opportunity for inference.
- Knowledge basis for inference engines.

The easy way to represent declarative facts is to use relations that can be represented as tables (as in databases). For example, information about the students in a department (Table 5.1). This table gives the information of three students who study in different classes with their marks for subject/total.

TABLE 5.1 Simple relational knowledge

Name	*Class*	*Marks*
C. Vinod	S3 CS	564
J. Prakash	S5 CS	430
R. Hari	S5 EC	390

Major problem is that there is not much knowledge represented by this method. So, we need some mechanisms to improve the representation. The inference engine obtains new knowledge from the representations.

Inheritable knowledge

We have seen that relational knowledge is made up of objects consisting of attributes and corresponding associated values. We can extend the base more by allowing property inheritance mechanisms. Elements that inherit values from members of a class and data organised into a hierarchy of classes is an inheritance property. A possibility occurs in the augment for basic representation with inference mechanism that operates on a structure of representation. This is effective when the designed structure is used by inference mechanism.

It is useful to structure the knowledge hierarchically (taxonomical hierarchy). The goal is to represent the knowledge as a graph or a tree and to use generalisation and specialisation goal (Figure 5.3). The nodes are concepts/classes, the edges are relations with is-a(class-class) relation or instance-of (class-instance) relation. The reasoning mechanism is the inheritance of properties and values of simple/multiple inheritance or default values.

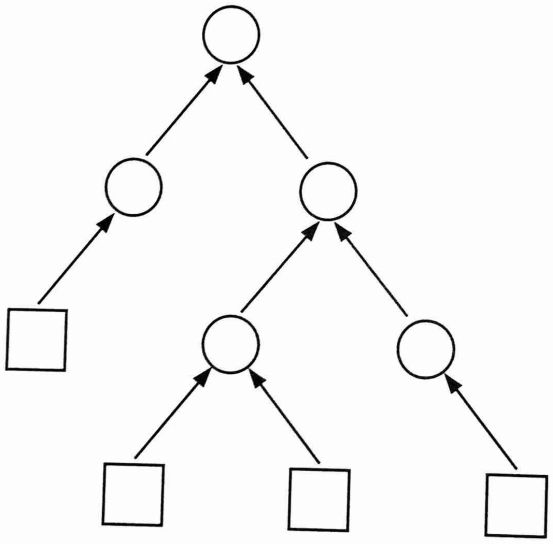

Figure 5.3 Inheritable knowledge.

Inferential knowledge

Inheritance is a powerful form of inference, but it is not the only useful form. Sometimes all the power of traditional logic is necessary to describe the inferences that are needed. The knowledge is described using logic. The semantics of the logic connectives and inference mechanisms (For example, Modus Ponens rule) can be used to obtain new knowledge

$$\forall\, x; y : \text{person}(x) \wedge \neg \text{under age}(x) \wedge \neg\, \text{job}(x; y) \rightarrow \text{unemployed}(x)$$

The inference mechanisms for first order logic are the different algorithms from automatic theorem proving. They are used for problem-solving that describes relationships between various ideas such as kind of, part of, and grouping of something. They also gives relationship that exists between concepts or objects.

5.2.2 Procedural Knowledge

Procedural knowledge can be represented as programs in many ways. The machine uses this knowledge when it executes the code to perform a task. This knowledge includes control information in its representation and/or specific procedures needed to use it. The knowledge is defined as algorithms that allow to infer new knowledge from facts. This kind of knowledge is computationally more efficient, but it is more difficult to obtain new knowledge from it and it is complex to acquire/modify.

For example, Date of birth = DD-MM-YYYY; function Age (Date of birth: date)

Procedural knowledge usage has some advantage as follows:

- Domain specific or heuristic can be represented.
- Facilitates extended logical inferences, such as default reasoning.
- Models a side effect of actions that are used in procedural knowledge. The generated rules may become false in time and it is tricky to keep track of this rule in large systems.

Some problems using procedural knowledge are based on certain procedure as follows:

1. Completeness: We cannot represent all cases.
2. Consistency: All deductions are not correct.
   For Example, If we know that *Tweety is a bird* we might deduce that *Tweety can fly*. Later we might discover that *Tweety is an emu.*
3. Modularity: Which is sacrificed. Changes in knowledgebase might have far-reaching effects.
4. Cumbersome: This is controlling information.

5.3 Knowledge Representation

In order to solve a complex problem in artificial intelligence, one needs both large amount of knowledge and some mechanism for manipulating that knowledge to create solution to new problems. A variety of ways to represent knowledge (facts) have been exploited in artificial intelligence programs.

First we need to finalise on the types of knowledge that need to be represented in artificial intelligence systems as follows:

Objects: Facts about the real-time entities in our world domain. For example, "Guitar has strings, trumpets are brass instruments".

Events: Actions that occur in our world. For example, "John played the guitar in 13AD's band".

Performance: It represents a behaviour. For example, "Playing the guitar", it involves knowledge about how to do it.

Meta-knowledge: It is the knowledge about a knowledge. This knowledge can read straight signs along the way to find out where it is. For example, "Jain's robot who plans a trip".

Thus, for problem solving in artificial intelligence, we need to represent knowledge. Two entities deals in the knowledge representation.

Facts: What we need to represent to the truths about the real-world. This can be regarded as the knowledge level for a domain. We have to manipulate the representation of the facts for a sentence or a knowledge. This is regarded as the symbol level since we usually define the representation in terms of symbols that can be translated by programs.

5.3.1 Approaches to Knowledge Representation

Basically four properties are needed to process a good system for the representation of knowledge in a particular domain.

1. **Representational adequacy:** Ability to represent all kinds of knowledge that are needed in a domain. It is how facts about the world can be represented in data structures. If representation is too weak, you cannot get at necessary discriminations in the representational adequacy.
   Consider the following facts:

   "John believes no-one likes Brussels sprout".

   "John will have to finish his assignment before he can start working on his project".

   These all can be represented as a string! But it is harder than to manipulate and draw conclusions.

2. **Inferential adequacy:** It is an ability to formulate the representational structures in such a way as to derive a new structure corresponding to new knowledge inferred from old. Representing knowledge is not very interesting unless you can use it to make inferences. That is, drawing new conclusions from existing facts.

   "If its raining, John never goes out"

   "It is raining today so."

   For this, come up with solutions to complex problems, using the represented knowledge. Inferential adequacy refers to how easy it is to draw inferences using represented knowledge. Representing everything as natural language strings has good representational adequacy and naturalness, but very poor inferential adequacy.

3. **Inferential efficiency:** Ability to incorporate knowledge structure with additional information that can be used to focus the attention of inference mechanism in most promising directions. You may be able to make complex deductions given knowledge represented in a sophisticated language.
   But it may be just too inefficient. Generally, more complex the possible deductions, the less efficient will be the reasoner which needs representation and inference system sufficient for the task, without being hopelessly inefficient.

4. **Acquisitional efficiency:** This is an ability to acquire new knowledge using automatic methods wherever possible rather than confidence on human intervention. It must allow to incorporate new knowledge to the representation easily. Ideally an intelligent agent should be able to obtain new information autonomously and incorporate it to the representation.

5.3.2 Issues in Knowledge Representation

A large amount of knowledge and some mechanism for manipulating that knowledge is the key of problem-solving. The basic goal of knowledge representation is to assist inference (conclusions) from knowledge. The issues arise while many knowledgebase techniques are used in a single problem. Some of these are explained as follows:

1. **Important attributes:** There are two attributes that are of very general significance. These attributes are important because they support property inheritance. There are two attributes "instance" and "isa", that are of general significance.
2. **Relationships among attributes:** The attributes that are used to describe objects as entities that we represent. There are four such properties as follows:
   (a) Inverse—When checking consistency, while a value is added to one attribute.
   (b) An 'isa' hierarchy of attribute—It is a generalisation vs specialisation. For example, height of a person is a specialisation of general attributes like physical size.
   (c) Techniques for reasoning about value—Reasoning values of attributes will not be given explicitly. Information values like height, weight, age, etc.
   (d) Single-valued attribute—It is a unique value that replaces knowledge. For example, president of India.
3. **Choosing the granularity of representation:** The level at which the knowledge needs to be represented and what are the primitives. Primitives are fundamental concepts such as seeing, playing and holding. English is a very rich language with over half a million words available in its dictionary. It is clear that we will find difficulty in deciding upon which words to choose as our primitives in a sequence of situations.
   There are quite a few arguments against the exercise of low-level primitives. First, a lot of storage when broken down into primitives is one of the simple high-level facts. Second, a related problem in knowledge is initially presented to the system in a moderately high-level form, such as in English, then considerable work must be done to reduce the knowledge into primitive form. Third, problems with the low-level primitives are in use in many domains. It is not at all clear what should be the primitives to be used.

   Example of Granularity:
   Suppose we are interested in following facts:
   "John spotted Maya"
   This could be represented as
   Spotted (agent(John), object (Maya))
   Such a representation would make it easy to answer questions such as:
   "Who spotted Maya?"
   Suppose we want to know:
   "Did John see Maya?"
   Given only one fact, we cannot discover that answer.
   We can add other facts, such as

Spotted (x, y) → saw (x, y)
We can now infer answer to the question.

4. **Representing sets of objects:** There are several reasons to represent a set of objects. First, there are some properties that are true for sets and that are not true for the individual members of the set. There are three obvious ways in which sets may be represented. The simplest way is 'name' representation. There are two ways to state a definition of a set and its elements. First, list members of the set. Such a specification is called an extensional definition. Second, is to provide a rule that when particular object is evaluated, return true or false depending on whether the object is a set or not.

   Example: Consider the assertion made in the sentences:

   "In Australia, there are more sheeps than people", and
   "English speakers are found all over the world."

   To describe these facts, the only way is to attach assertion to the sets representing people, sheep, and English.

5. **Find the right structure:** For describing the right structure of a particular situation, we can form certain requisites. This requires, selecting an initial structure and then revising the choice. It is necessary to solve the following problems for doing the above requirements:

   - How to carry out a preliminary selection of the most suitable structure?
   - How to fill in suitable details from the present situations?
   - How to find an improved structure if one chosen at first turns out not to be suitable?
   - What to do if none of the accessible structures is suitable?
   - When to produce and keep in mind a new structure?

   There is no good, general purpose method for solving all these problems. Some knowledge representation techniques solve some of them. It is very important in the structure of a sentence. Consider the following example:

   "John went to Ambrozia last night. He ordered a large rare pizza. He paid the bill and left".
   "Did John eat dinner last night?"

   The answer is 'yes', for the above question. Notice that nowhere in the story telling John is eating. The knowledge representation solves such problems.

5.4 Knowledgebase

A knowledgebase is a type of database for knowledge management. The knowledgebase contains means for information to be collected, organised, shared, searched and utilized. The knowledgebase has facts, rules and its connected relations. The facts are specifics to the world. For example, Consider the following facts:

"VIT is a deemed University"

"The first thing I did at the party was talk to John".
The rules (axioms) in the knowledgebase that describe ways to infer new facts from existing facts. For example, consider the following sentences:

"All triangles have three sides"
"All elephants are grey"

There are general facts which are known to all. Such facts are stored in knowledgebase. The facts and rules are stated in a formal language by which it can connect each other when problem-solving encounters. This is generally some form of logic (predicate calculus). For a practical side, in order to perform their desired tasks, medical experts require access to knowledge about their domains. Figure 5.4 shows an example of relationship—facts and representation schemes with respect to knowledgebase.

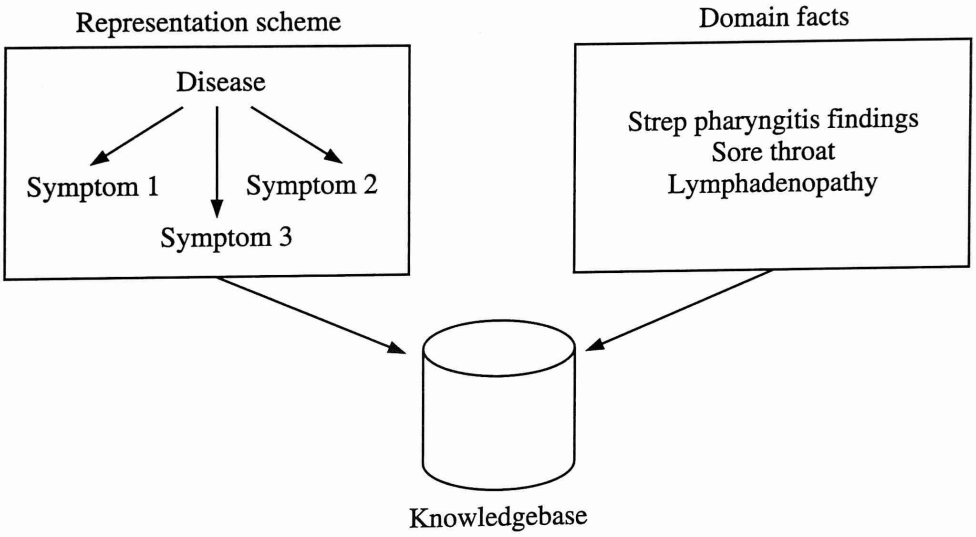

Figure 5.4 Knowledgebase representation—Medical facts.

There are mainly two types of knowledgebases that are essentially closed or open information repositories as follows:

Machine-readable knowledgebases: These store knowledge in the form of a computer readable attributes. Automated deductive reasoning is the major concern used in this storage. This representation contains a set of data, often in the form of rules that describe the knowledge in a logically consistent manner. Logical operators (conjunction, disjunction), implication and negation are used to build relationship between bricks of facts.

Human-readable knowledgebases: These are designed to allow people to retrieve and use the knowledge. These are implemented in the form of helpdesks that share information among people within a domain. They may store troubleshooting information, articles, user manuals, knowledge tags, white papers, or answers to regularly asked questions. A web-search engine can be used to locate information in the system, or users may browse through a scheme of classification.

6.1 First Order Logic

First order logic is a collection of formal systems used in Computer science, Mathematics, Philosophy and Linguistics. First order logic is one of the knowledge representation methods in artificial intelligence. It is an extension to propositional logic which communicates to represent the natural language statements in a short way. Consider the following sentence:

"John is my friend"

Here 'John' is the subject and 'is my friend' is the predicate.

Consider two facts as follows:

'P' represents the fact "John likes chocolate"

'Q' represents the fact "John has chocolate"

How will you represent "John likes chocolate and he does not have any" in simple form?

Simply, $P \wedge \neg Q$

So, what is the meaning of $P \rightarrow Q$?

"If John likes chocolate then John has chocolate". Is it correct meaning of the symbolic representation?

It is a very general system of logic which accurately expresses a huge variety of assertions and modes of reasoning. Any sentence in a language is explained with syntax and semantics. To explain syntax we need to look the legitimate expressions. The legitimate expression in predicate calculus is known as well-formed formula (*wff*). The components of an expression are predicate, symbols, objects, variable symbols, function symbols and constant symbols.

- Examples of objects are people, houses, numbers, theories, colours, baseball game etc.

- Examples of relations are an ordered pair, for example, brother of, bigger than, inside, part of, etc.
- Examples of properties are round, prime, multistoried, etc.
- Examples of functions are best friend, father of, mother of, etc.

User defines these primitives:

- Constant symbols (That is, the 'individual entities' in the world)
  Example, John, 3
- Function symbols (mapping between individuals)
  Example, father-of(Reji) = John, colour-of(moon) = White
- Predicate symbols (mapping from individuals to truth values)
  Example, greater(5,3), white(moon), colour(moon, white)
- A predicate is a relation which has objects as its arguments.
  Example, "John likes football".
  This sentence can be represented as: likes(John, football)

In predicate calculus, certain connectives and quantifiers are used for creating *wff*. These symbolic representations are the following:

Connectives

$\wedge$	and
$\vee$	or
$\neg$ or $\sim$	not
$\rightarrow$	implication
$\Leftrightarrow$	iff

Quantifiers

First order logic contains two standard quantifiers, called Universal and Existential.

Universal quantifiers

Universal quantification corresponds to conjunction ('and') in which $(\forall x)P(x)$ means that P holds for all values of x in the domain associated with that variable.

Example, $\forall x$: penguin(x) $\rightarrow$ birds(x) means "All penguins are birds".

Universal quantifiers are usually form 'if-then rules' by using 'implies' symbol.

Example, $\forall x$: computer-science-student(x) $\rightarrow$ smart(x) means "All computer science students are smart."

It is rarely used universal quantification to make cover statements about every individual in the world such as:

$\forall x$: computer-science-student(x) $\wedge$ smart(x) meaning that "everyone in the world is a computer science student and is smart".

Existential quantifiers

Existential quantification corresponds to disjunction ("or") in which $(\forall x)P(x)$ means that P holds for some value of x in the domain associated with that variable.
Example, $\exists x$: bird(x) $\wedge$ lives-Antarctica(x)

Existential quantifiers are usually used with "and" to specify a list of properties or facts about an individual.
Example, $\exists x$: computer-science-student(x) $\wedge$ smart(x) means "there is a computer-science student who is smart".

A common mistake is to represent this English sentence as the first order logic sentence: $\exists x$: computer-science-student(x) $\rightarrow$ smart(x)

But consider what happens when there is a person who is NOT a computer-science-student.

In predicates, the coefficient having connective quantifiers, true and false values form the basis of predicate calculus. Sentences are built up from terms and atoms. Table 6.1 gives priority based operations of symbols and connectives.

TABLE 6.1 Connectives and symbols in decreasing order of operation priority

Connective	*Symbols*	*Read as*
Assertion	p	"p is true"
Negation	$\neg$p	"p is false"
Conjunction	p $\wedge$ q	"both p and q are true"
Disjunction	p $\vee$ q	"either p is true, or q is true, or both "
Implication	p $\rightarrow$ q	"if p is true, then q is true", "p implies q"
Equivalence	p $\leftrightarrow$ q	if and only if "p and q are either both true or both false"

A term which denotes a real-world individual can be a constant or variable symbol. It can also be an *n*-place function with n terms. For example, x and $f(x_1, ..., x_n)$ are terms, where each x_i is a term.

An atom is an *n*-place predicate of *n* terms. It can have value of either true or false. If P and Q are atoms, then $\neg$P, P $\vee$ Q, P $\wedge$ Q, P $\rightarrow$ Q, P $\Leftrightarrow$ Q are atoms.

If a sentence does not contain any "free variables" then it is termed as a well-formed *wff*. That is, all variables are either "bound" by universal or existential quantifiers. For example, $\forall x$: P(x, y) has x bound as a universally quantified variable, but y is free.

6.1.1 Basic Predicate Representations

Convert the following sentences into first order logic:

1. John lives in a yellow house.
   lives(John, house) ∧ colour(house, yellow)
2. John plays centre field or shortstop.
   plays (John, centre-field) ∨ plays(John, short-stop)
3. If the car belongs to John then it is green.
   belongs(John, car) → colour (car, green)
4. Voltaire did not write computer chess.
   ¬ write(voltaire, computer-chess)
5. All elephants are gray.
   ∀x: elephant(x) → colour(x, gray)
6. There is a person who wrote computer chess.
   ∃x person(x) ∧ wrote(x, Computer chess)
7. Every gardener likes the sun.
   ∀x: gardener(x) → likes(x, sun)
8. Not every gardener likes the sun.
   ¬ (∀x: gardener(x) → likes(x, sun))
9. You can fool some of the people all of the time.
   ∃x: (person(x) ∧∀ t(time(t) → can-fool(x, t)))
10. You can fool all of the people some of the time.
    ∀x (person(x) → ∀t (time(t) ∧ can-fool(x, t)))
11. You cannot fool all of the people all of the time.
    ¬ (∀x:∀t: (person(x) ∧ time(t) → can-be-fooled(x, t)))
12. All violet mushrooms are healthy.
    ∀x: (mushroom(x) ∧ purple(x)) → healthy(x)
13. No violet mushroom is healthy.
    ¬∃x: violet(x) ∧ mushroom(x) ∧ healthy(x)
    or, equivalently,
    ∀x: (mushroom(x) ∧ violet(x)) → ¬healthy(x)
14. There are exactly two violet mushrooms.
    ∃x: ∃y: mushroom(x) ∧ violet(x) ∧ mushroom(y) ∧ violet(y) ∧¬(x = y) ∧ (∀z) (mushroom(z) ∧ violet(z)) → ((x = z) ∨ (y = z))
15. John is not tall.
    ¬tall(John)
16. X is above Y if X is directly on top of Y or else there is a pile of one or more other objects directly on top of one another starting with X and ending with Y.
    ∀x: ∀y: above(x, y) ⇔ (on(x, y) ∨ (∃z) (on(x, z) ∧ above(z, y)))

6.1.2 Conversion of WFF to Clause Form

Clause form represents the logical database as a set of disjunctions of literals. A literal is an atomic expression or the negation of an atomic expression.

Algorithm

With the following example, we can convert a *wff* into the equivalent clause form:

$\forall x$: $\{P(x) \rightarrow \{\forall y[P(y) \rightarrow P(f(x, y))] \wedge \neg \forall y[Q(x, y) \rightarrow P(y)]\}\}$

1. Eliminate implication symbol using A → B by ¬AVB.

   $\forall x:\{\neg P(x) \vee \{\forall y[\neg P(y) \vee P(f(x, y))] \wedge \neg \forall y[\neg Q(x, y) \vee P(y)]\}\}$

2. Reduce the scope of each negation symbol by distributive laws (deMorgan's laws).

   $\forall x:\{\neg P(x) \vee \{\forall y[\neg P(y) \vee P(f(x, y))] \wedge \exists y[\neg Q(x, y) \vee P(y)]\}\}$

3. Standardise variables so that each quantifier binds a unique variable.

   $\forall x:\{\neg P(x) \vee \{\forall y[\neg P(y) \vee P(f(x, y))] \wedge \exists w[\neg Q(x, w) \vee P(w)]\}\}$

4. Move all quantifiers to the left of the formula without changing the relative order.

   $\forall x:\forall y \exists w:\{\neg P(x) \vee \{[\neg P(y) \vee P(f(x, y))] \wedge [\neg Q(x, w) \vee P(w)]\}\}$

5. Eliminate the existential quantifiers by Skolem constants.
   Every existentially quantified variable can be replaced by a unique Skolem function whose arguments are all the universally quantified variables on which the existential depends, without changing first order logic.

   Examples

   "Everybody likes something"

   $\forall x \exists y$: [Person(x) $\wedge$ Likes(x, y)]
   $\forall x$: [Person(x) $\wedge$ Likes(x, $S_1(x)$)]
   Where $S_1(x)$ = "that which x likes"

   Another example

   "Every philosopher writes at least one book"

   $\forall x$: $\exists(y)$[Philosopher(x) $\wedge$ Book(y)) $\rightarrow$ Write(x, y)]
   $\forall x$: [(Philosopher(x) $\wedge$ Book($S_2(x)$)) $\rightarrow$ Write(x, $S_2(x)$)]

   In our problem, after eliminating existential quantifier
   $\forall x:\forall y:\{\neg P(x) \vee \{[\neg P(y) \vee P(f(x, y))] \wedge [\neg Q(x, g(x)) \vee P(g(x))]\}\}$

6. Drop the prefix

   $\{\neg P(x) \vee \{[\neg P(y) \vee P(f(x, y))] \wedge [\neg Q(x, g(x)) \vee P(g(x))]\}\}$

7. Convert the matrix into a conjunction of disjunct.

   $(a \wedge b) \vee c \equiv (a \vee c) \wedge (b \vee c)$

   $\{\neg P(x) \vee \neg P(y) \vee P(f(x, y))\} \wedge \{\neg P(x) \vee \neg Q(x, g(x)) \vee P(g(x))\}$

8. Create a separate corresponding clause for each conjunct.
   (a) $\neg P(x) \vee \neg P(y) \vee P(f(x, y))$
   (b) $\neg P(x) \vee \neg Q(x, g(x)) \vee P(g(x))$
9. Standardise a part of the variable in the set of clauses generated in the previous step, that is, rename the variables so that no two clauses may be reference to the same variable.
   (a) $\neg P(x_1) \vee \neg P(y) \vee P(f(x_1, y))$
   (b) $\neg P(x_2) \vee \neg Q(x_2, g(x_2)) \vee P(g(x_2))$

6.1.3 Resolution

Resolution is a technique for proving theorems in propositional or predicate calculus that has been a part of problem-solving. Resolution proves a theorem by negating statement to be proved and adding this negated goal to a set of axioms that are known to be true. The steps involved in resolution technique are given in Algorithm 13.

Algorithm 13 Propositional resolution

Convert all the propositions of P to clause form.
Negate P and convert the result to clause form. Add it to the set of clauses obtained in step 1.
repeat
(a) Select two clauses and call their parent clauses.
(b) Resolve them together, then the resolvent will be the disjunction of all of the literals of both of the parent clauses with the following exception: if there are any pairs of literals L and $\neg$ L such that one of the parent clauses contains L and the other contains $\neg$L, then select one such pair and eliminate both L and $\neg$L from the resolvent.
If the resolvent is the empty clause, then a contradiction has been found.
If it is not found, then add in the set of clauses available to the procedure.
until either a non-progress condition or a contradiction is found

Example 6.1: Consider the following set of axioms in proportional logic.

1. P
2. $(P \wedge Q) \rightarrow R$
3. $(S \vee T) \rightarrow Q$
4. T

Corresponding clause form is

1. P
2. $\neg P \vee \neg Q \vee R$

3. ¬ S ∨ Q
4. ¬ T ∨ Q
5. T

Using resolution by refutation we can solve R (Figure 6.2). In the principle of resolution by refutation we negate the goal sentence and try to look at the given set of sentences for a match. That is, a predicate which is existing in the goal sentence. The matching will take place between predicates of the form as follows:

L and ¬ L

After matching, the variables will attain constant values from either of the predicates. This process is known as substitution. When a substitution occurs within the predicate of the form L and ¬ L, we say that unification has been carried out within L and ¬ L.

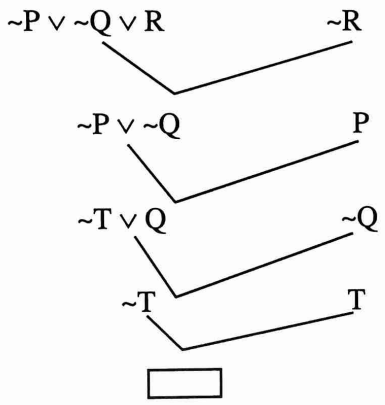

Figure 6.2 Refutation tree.

Resolution examples

1. Construct the following sentences into clause form and extract the answer.
   "If fido goes wherever John goes and if John is at school, where is fido?"
   The equivalent *wff*

   (a) ∀x: at (John, x) → ¬ at (fido, x)

   (b) at (John, school)

   Goal: at (fido, x)
   The equivalent clause form is

   (a) ¬ at (John, x) ∨ at (fido, x)
   (b) at(John, school)

   By resolution, we need negation of the goal statement ¬ at (fido, x). To carry out answer extraction process we shall consider both the goal and the negation. Resolution tree is shown in Figure 6.3.

2. Convert the sentences into clause form and answer the question:
   Tony, Tensing and Hilari belong to the MountainClub. Every member of the MountainClub is either a skier or a mountain climber or both. No mountain climber

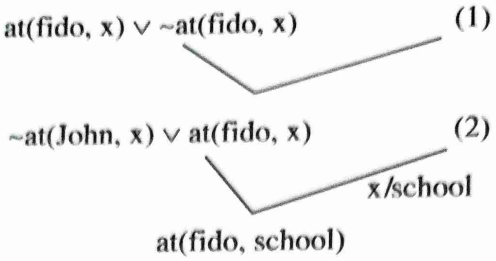

Figure 6.3 Resolution tree.

likes rain, and all skiers like snow. Ellen dislikes whatever Tony likes and likes whatever Tony dislikes. Tony likes rain and snow.

Question: Is there a member of the MountainClub who is a mountain climber but not a skier?

Translation into First Order Logic (FOL)

We can now translate the above English sentences into the following FOL *wffs*:

(a) $\forall x$: Skier(x) $\vee$ mountain-climber(x)
(b) $\neg\ \exists x$: mountain-climber(x) $\wedge$ likes(x, Rain)
(c) $\forall x$: skier(x) $\rightarrow$ likes(x, Snow)
(d) $\forall y$: likes(Ellen, y) $\Leftrightarrow$ $\neg$likes(Tony, y)
(e) likes(Tony, Rain)
(f) likes(Tony, Snow)
(g) Query: $\forall x$ mountain-climber(x) $\wedge$ $\neg$skier(x)
(h) Negation of the Query: $\neg\exists x$ mountain-climber(x) $\wedge$ $\neg$skier(x)

Conversion to clause form

(a) skier(x1)$\vee$ mountain-climber(x1)
(b) $\neg$mountain-climber(x2) $\vee$ $\neg$likes(x2, Rain)
(c) $\neg$S(x3) $\vee$ likes(x3, Snow)
(d) $\neg$likes(Tony, x4) $\vee$ $\neg$likes(Hilari, x4)
(e) likes(Tony, x5) $\vee$ likes(Hilari, x5)
(f) likes(Tony, Rain)
(g) likes(Tony, Snow)
(h) Negation of the Query: $\neg$mountain-climber(x7) $\vee$ skier(x7)

Resolution refutation proof

Clause 1	Clause 2	Resolvent	
8	1	9. skier(x1)	x7/x1
9	3	10. likes(x1, Snow)	x3/x1
10	4	11. $\neg$likes(Tony, Snow)	x4/Snow, x1/Hilari
11	7	12. False	{}

Answer extraction:
The refutation tree is shown in Figure 6.4.

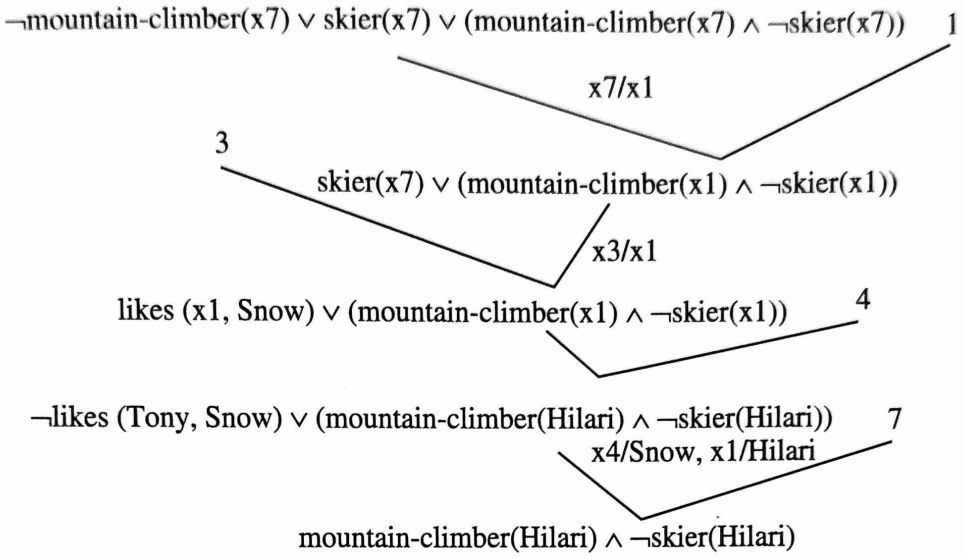

Figure 6.4 Refutation tree.

Answer to the Query:
Hilari

3. Using resolution by refutation answer the goal sentence.

   (a) Whoever can read is a literate.
   (b) Dolphins are not literate.
   (c) Some dolphins are intelligent.

   Goal sentence: 'Some who are intelligent can not read' Solution:

   Classical form can be

   R : Read L : Literate I : Intelligent D : Dolphin

Then wff:	Clause form
$R(x) \rightarrow L(x)$	1. $\neg R(x1) \vee L(x1)$
$D(x) \rightarrow \neg L(x)$	2. $\neg D(x2) \vee \neg L(x2)$
$D(x) \rightarrow I(x)$	3. (a) D(x3) (b) I(x4)

   Goal is: ¬I(x5) vee R(x5)

   Negation of goal sentence is wrong. So, goal sentence will be true, that is, some who are intelligent can not read. The refutation tree is given in Figure 6.5.

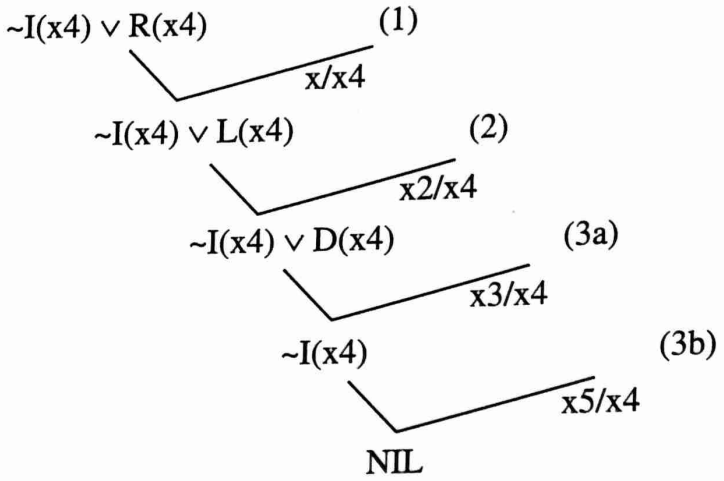

Figure 6.5 Refutation tree.

6.1.4 Issues with Resolution

A question arises, shall we use resolution to answer most of the questions? Sometimes clause forms result problems in their meaning when we convert sentence in first order logic to clause form. The following are certain issues with resolution techniques:

- Requires full formal representation in first order logic (for conversion to clause form).
- Resolution defines a search space (which clauses will be resolved against which others define the operators in the space) → search method required.
- In the worst case, resolution is exponential in the number of clauses to resolve. Actual: exponential in average resolvable set (= branching factor).
- Can we define heuristics to guide search for Best first search, or A* or B*? (Not in the general case)?

Algorithm 14 Unification algorithm

Function unify (E_1, E_2)
Case:
1. Both E_1 and E_2 are constant or the empty list
   if $E_1 = E_2$ then return
   Else **return** FAIL
2. E_1 is a variable:
   If E_1 occurs in E_2 then return FAIL
   else return $\{E_1/E_2\}$
3. E_2 is a variable:
   if E_2 occurs in E_1 then return FAIL
   else return $\{E_1/E_2\}$
4. Either E_1 or E_2 are empty then return FAIL

Otherwise:
$H\ E_1$ = First element of E_1
$H\ E_2$ = First element of E_2
$SU\ BS_1$ = unify ($H\ E_1$, $H\ E_2$)
if $SU\ BS_1$ = FAIL then return FAIL
$T\ E_1$ = apply ($SU\ BS_1$, rest of E_1)
$T\ E_2$ = apply ($SU\ BS_1$, rest of E_2)
$SU\ BS_2$ = unify ($T\ E_1$, $T\ E_2$)
if $SU\ BS_2$ = FAIL then return FAIL;
else return composition ($SU\ BS_1$, $SU\ BS_2$)

For example:
Unify ((parents x (father x) (mother bill)), (parents bill (father bill) y))
The step by step illustration of unification algorithm is given in Figure 6.6.

6.3 Frames

Frames were introduced by Mervin Minsky in 1975, as a data structure to represent a mental model of a stereotyped situation, such as, driving a car, attending a meeting, eating in a restaurant, etc. Knowledge about an item is stored in a single unit called a frame. When a new situation is encountered, an appropriate frame is selected for reasoning to the situation.

Frames are general record structures which consist of a collection of slots and slot values. Slots contain names, values and such fields called links. The links may also have names and any number of values. The slots in a frame specify a character of an entity for which the frame is being represented. Some slots will also include instructions for manipulations like entering default values, a few conditions for filling a slot for procedures such as:

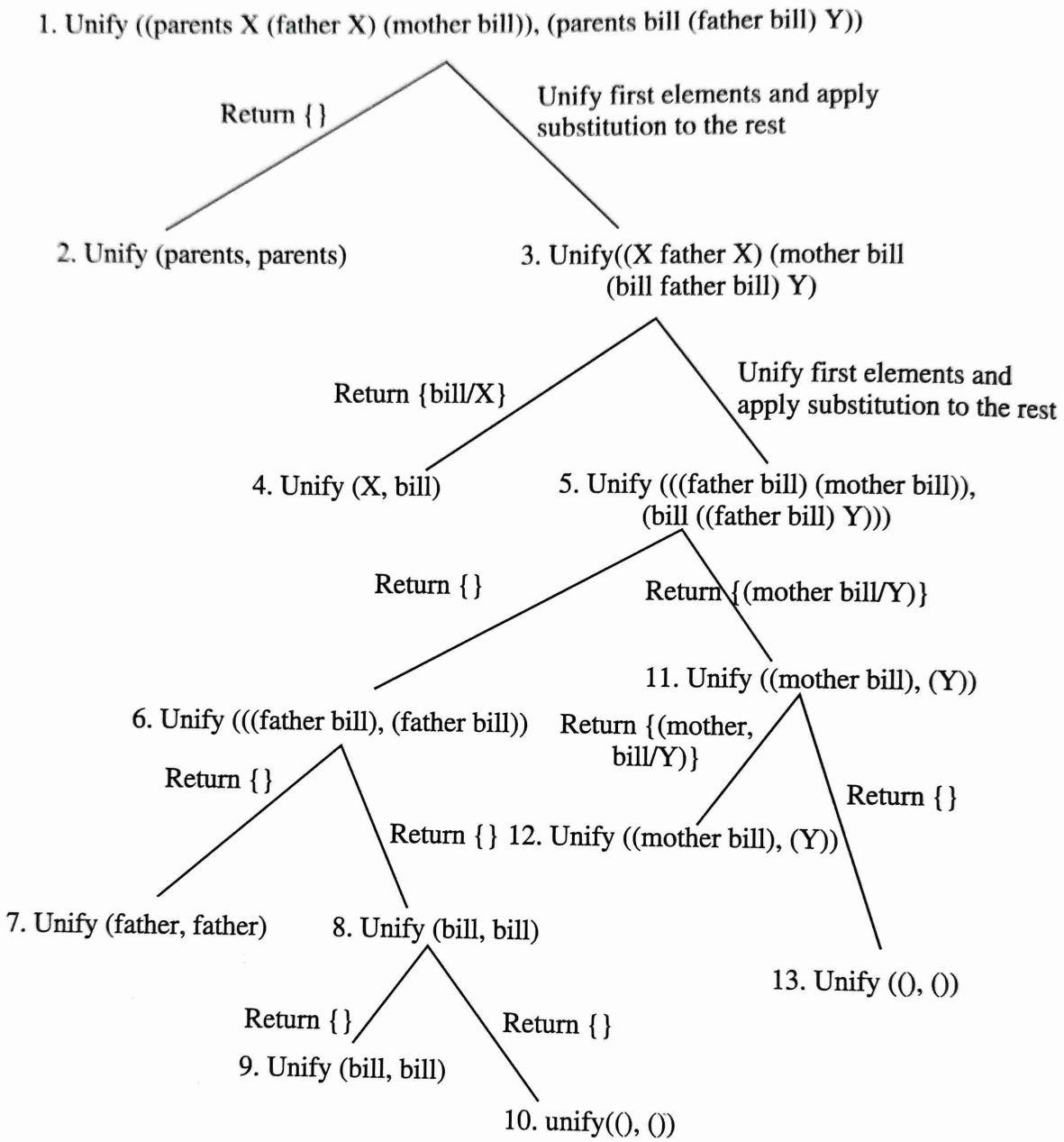

Figure 6.6 Trace of unify ((parents X (father X) (mother bill)) and (parents bill (father bill) Y).

If needed, If added, If removed, etc.

The slots are similar to traditional record that contains information relevant to stereotyped entities. The slot in the frame contains information such as:

- Frame identification
- Relationship of this frame with other frames
- Descriptors of requirements for a frame
- Procedural information used in the structure described.
- New instance information

Frame system supports class inheritance. The slots and default values of a class frame are inverted across the class/subclass and class/member hierarchy.

Example: A frame that describes lecturer in Operating system by Prof. John is shown in Figure 6.7.

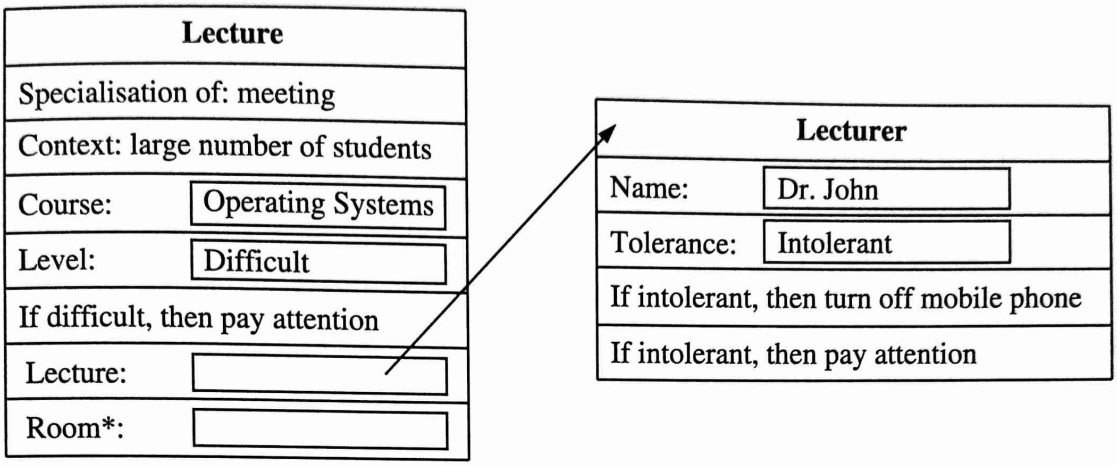

Figure 6.7 Frame representation—lecturer.

Example: A frame description of a hotel room is shown in Figure 6.8. This frame describes specialisation of a hotel room which has a bed with king size and other furniture in that room.

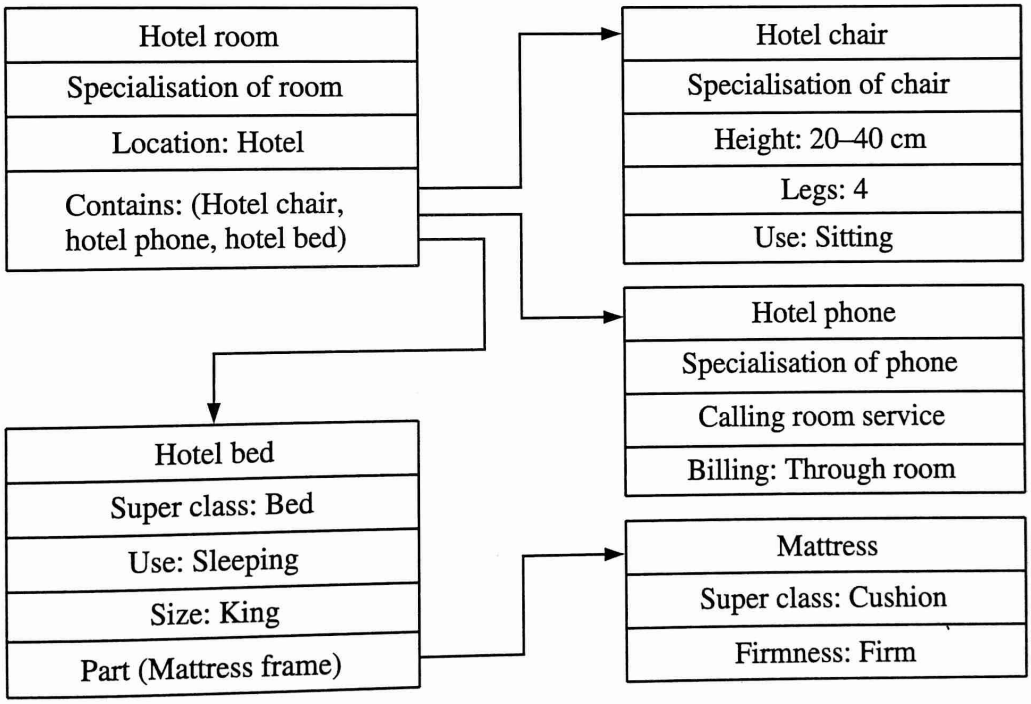

Figure 6.8 Frame representation for a hotel room.

Another example of a frame 'animal' is given in Figure 6.9. Here we have slot-value (elephant, size, large), instance (Hari, elephant), value (obj, slot, V) : instance (obj, class), slot-value (class, slot, V). Frames also allow multiple inheritance (Hari is an elephant and is a circus animal). Another example is given in Figure 6.10 regarding vehicle frame.

Mammal:
  Subclass: Animal

Elephant:
  Subclass: Mammal
  Size: Large
  Haspart: Trunk

Hari:
  Instance: Elephant
  Likes: Apples

Figure 6.9 Frame representation—animal.

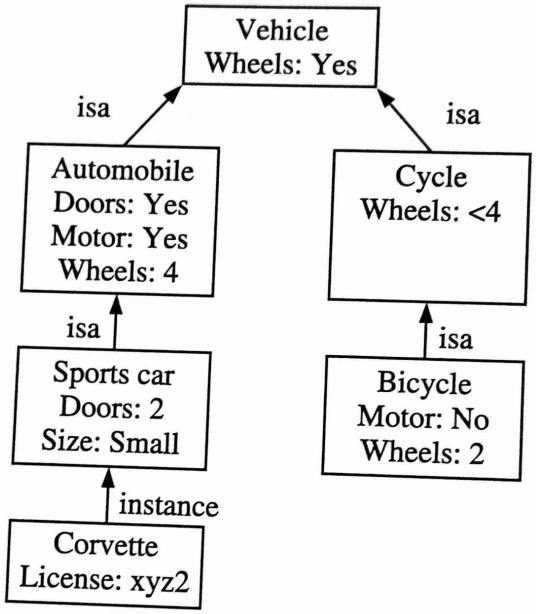

Figure 6.10 Frame representation–vehicle.

Advantages of frames

- A frame collects information about an object in a single place in an organised fashion.
- By relating slots to other kinds of frames, a frame can represent typical structures involving an object; these can be very important for reasoning based on limited information.
- Frames provide a way of associating knowledge with objects (via the slot procedures).
- Frames may be a relatively efficient way of implementing artificial intelligence applications (direct procedure invocation versus search in a logic system).
- Frames allow data that are stored and computed to be treated in a uniform manner. (Example, AGE might be stored, or might be computed from BIRTHDAY.)

Object-oriented programming has much in common with frames. Frames are good for applications in which the structure of the data and the available information are relatively fixed. The procedures (methods) associated with slots provide a good way to associate knowledge with a class of objects. Frames are not particularly good for applications in which deep deductions are made from a few principles (as in theorem proving).

Disadvantages of frames

Frames encourage baroque representations; little guide to good structuring of a domain. All problems with respect to the object-oriented systems are also applicable for frames.

Some things that can be represented in logic cannot be represented well, or at all, in frames.

- Slot fillers must be "real" data. For example, it is not possible to say that John is a butcher or a baker, since there is no way to deal with a disjunction in a slot filler.
- It is not possible to quantify over slots. For example, there is no way to represent "Some student made 100 on the exam".
- It is necessary to repeat the same information to make it usable from different viewpoints, since methods are associated with slots or particular object types. For example, if there are knowledge stores regarding John, it may be easy to answer "whom does John love?" but hard to answer "who loves Mary?".

In summary, frames are instance of structured representation (schemes). Frames posses static data-structure representing stereotyped situation. Frames provide object-oriented system properties. Frames offer which things were just typical of a class, and which were definitional for knowledge representation and could not be overridden. Using an *asterix* (*) to denote typical values.

6.4 Conceptual Dependency

It is a theory of how to represent a kind of knowledge about event, that is, usually contained in natural sentences. The goal is to represent the knowledge in a way that facilitates drawing inference from the sentences and which is independent of the language in which the sentences were originally stated.

The Conceptual Dependency (CD) theory was proposed by Schank and Ridge in 1973, offering a set of four primitive conceptualisations from which the world of meaning is built. They are equal and independent.

1. ACTs actions
2. PPs objects (Picture producers)
3. AAs modifiers of action (Action aiders)
4. PAs modifiers of objects (Picture aiders)
5. T (Times)
6. LOC (Locations)

A primitive is a set of idea (words) that maps to an object. There are some primitives as follows:

1. ATRANS—Transfer a relationship (example, give)
2. PTRANS—Transfer physical location of an object (example, go)
3. MOVE—Move body part by owner (example, kick)
4. ATTEND—Focus sense organ (example, listen)

5. MTRANS—Transfer mental information (example, tell)
6. SPEAK—Utter a sound (example, say)
7. GRASP—Actor grasping a thing (example, clutch)
8. INGEST—Actor ingesting a thing (example, eat)
9. EXPEL—Actor getting rid of a thing from body
10. PROPEL—Apply physical force to an object (example, push)

Conceptual dependency possesses certain symbols, which gives interconnection between two objects. These symbols aid relations between each objects. The symbols used in CD are as follows:

1. ←: Indicates the direction of dependency
2. ⇔: Indicates the agent-verb relationship
3. P Indicates the past tense
4. O Indicates the object relation
5. D Indicates the direction of the object in the action
6. C Indicates the condition

Example:

Consider the following sentence:

"John throws the ball"

$$\text{John} \Leftrightarrow \text{PROPEL} \overset{O}{\leftarrow} \text{Ball}$$

Conceptual dependency offers a number of important benefits. By proving a formal theory of natural language semantics, it reduces problems of ambiguity. Second, the representation itself directly captures much of natural language semantics, by attempting to provide a canonical form for the meaning of sentences.

For example: "John prevented Mary from giving book to bill".

The CD representation for this sentence is shown in Figure 6.11.

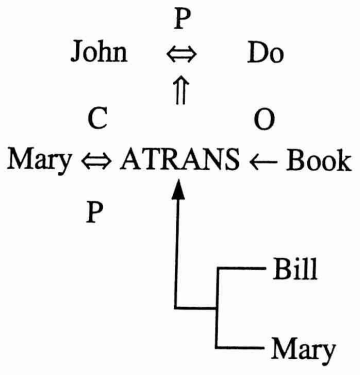

Figure 6.11 CD representation.

Advantages of CD

- Usage of these primitives involves fewer inference rules.

- CD structures already have many of the inference rules.
- The holes in the initial structure help to establish points which are not yet focussed.

Disadvantages of CD

- Knowledge needs to be decomposed to low-level primitives.
- It is always a tedious task to find the correct set of primitives.
- Many inferences are required.
- Even the representation of a simple action can become complex.

6.5 Scripts

A script is a structured representation describing a stereotyped sequence of events in a particular context. The script was originally designed by Schank in 1977. The knowledgebase representation in terms of the situations that the system is supposed to understand.
They are extremely beneficial because of the reasons as follows:

- The events always tend to occur in known patterns or runs.
- There is always a relationship between the existing events.
- There exists an entry condition(s) which allows event to take place.
- There are prerequisites for the events that are taking place. For example, when a student progresses through a degree scheme or when a customer buys a product.

The components of scripts are as follows:

- **Entry conditions** include open conditions, satisfied before the events described.
- **Result** or 'fact' is true once the script has terminated.
- **Props** or the 'things' that support the content of a script. The set of props supports reasonable default assumptions about a situation.
- **Roles** are actions that an individual participant performs.
- **Tracks** are the specific variation on a more general pattern that is represented by a particular script.
- **Scenes** are the actual sequence of events that occur.

The events described in a script form a giant causal chain. A set of entry conditions are enabled in beginning of the chain which enables the first event of the script to occur. The end of the chain is the set of results which may enable later events or event sequences to occur. Within the chain, events are connected to earlier events that make them possible, and to later events that they enable.

Example: A railway ticket reservation script

Script : Railway station
Track : Ticket booking
Props : Counters

D = Display board with information
F = Ticket
Credit card
Money

Roles:

S = Traveller
W = Person on the ticket counter (Current tickets)
C = Person issues Tatkal reservation tickets
M = Ticket examiner
O = Railway management

Every condition:

S required a ticket
S has money

Result:

S has less money
O has more money
S needs not reservation
S is pleased (optional)

Scene 1: Entering

S PTRANS S into station
S ATTEND eyes to counter
S MBUILD where to take
S PTRANS S to counter
S MOVE S to ticket counter

Scene 2: Reserving

(D on his hand) (W points out D) (S asked for D)

S PTRANS Menu to S

↓

S MTRANS signal to W
W PTRANS W to counter
S MTRANS need D to W
W PTRANS W to D
W PTRANS W to counter ↙

*S MBUILD Choice of F
S MTRANS signal to W
W PTRANS W to counter
S MTRANS 'I want F' to W
W MTRANS 'Tatkal is available' to S
S MTRANS 'I want Tatkal F' to W
W PTRANS W to C

W MTRANS (A TRANS F) to C

C MTRANS 'Print F' to W
W PRANS W to S
W MTRANS 'Print F' to S
(go back to *)

C DO (prepare F script)
to scene 3

Scene 3: Checking
C ATRANS F to W
W ATRANS F to S
S INGEST F
(Option: returns to scene 2 to more tickets, otherwise go to scene 4)

Scene 4: Leaving
(W ATRANS money to S)
W PTRANS W to S
W ATRANS credit card to S
S PTRANS S to M
S ATRANS money to M
S PTRANS S moves out of the station

S MTRANS to W
W Move (give credit card)

Some additional points to note on scripts:

- For applying a script, it needs to be activated, and the activation depends on the significance of the particular script.
- If a subject is noted in passing then a pointer to that script could be held.
- For important topics the script should be opened.
- It is not desirable in many active scripts that might be like having too many windows open on the display or too many recursive calls in a program.
- For events following known trail, we can use active scripts to represent the actions involved and use them to answer detailed questions.
- Different trails may be allowed for different outcomes of scripts (For example, the restaurant script).

Advantages

- Ability to predict events.
- A single coherent interpretation may be built up from a collection of observations.

Disadvantages

- It is not as general as frames.
- It is not suitable to represent all kinds of knowledge.

6.6 Semantic Network

A semantic network or net is a graphic notation for representing knowledge in patterns of interconnected nodes and arcs. The primary version of semantic networks was used in areas of philosophy, psychology and linguistics but now computer implementations of semantic networks are developed for artificial intelligence and machine translation. It was Collins and Quillian who developed semantic networks during their research on human information storage and response times.

A semantic network representation is similar to network model of database management system. That is, entities are connected by relationship. It is understood that semantic network representation gives us clustering of several units namely entities through relationship. Network representation is necessary because many of the English sentences cannot be exactly represented by predicate calculus. A few of the relationships used in semantic nets are:

isa
a kind of - AKO
Instance-of F
Member-of
Has-parts

An example is given in Figure 6.12. In this example, Tweety is a yellow bird which has wings and can fly. This statement gives a network architecture. A semantic network to represent birds is shown in Figure 6.13.

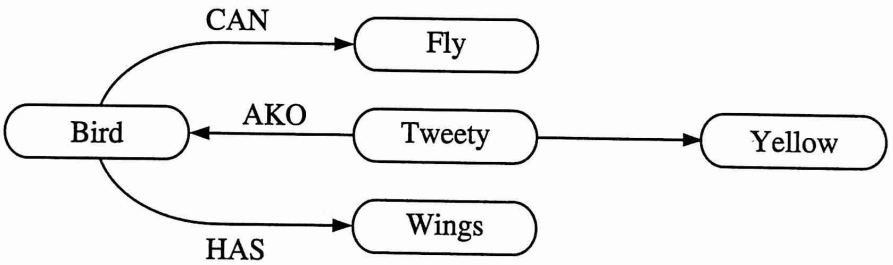

Figure 6.12 A simple semantic network.

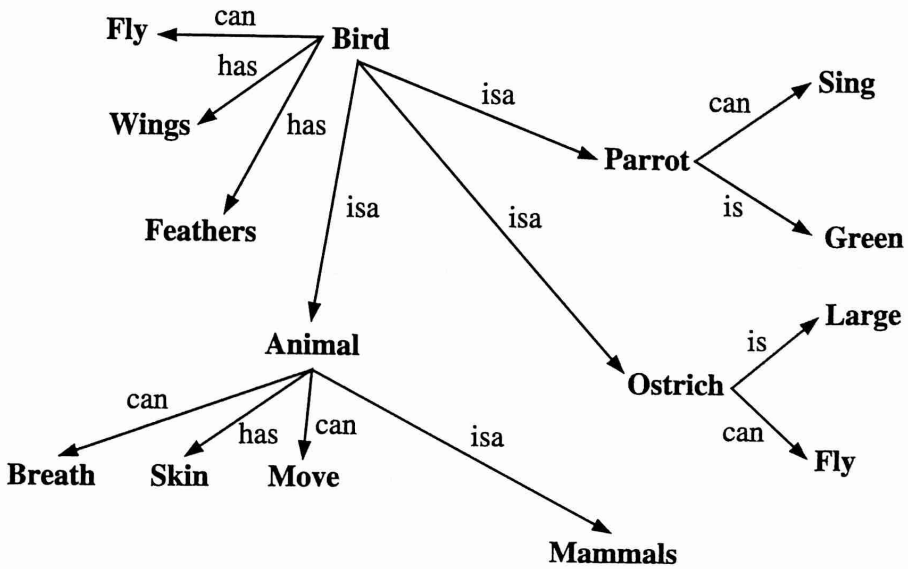

Figure 6.13 A semantic network—birds.

For all semantic networks there is a declarative graphic representation which is used for representing knowledge or to support automated systems for reasoning about knowledge. Many versions are highly informal, but there are versions which are formally defined by systems of logic. The below mentioned are the most common types of semantic networks.

1. **Definitional networks** highlight the subtype or 'isa' relation between a concept type and a newly defined subtype. This network is called a generalisation network hierarchy. There are rules of inheritance that give support for copying properties defined for a super-type to all of its subtypes. The assumed information in these networks is true by its definition and operates in semantic architecture.
2. From the asset propositions, we can design **assertional networks**. It assumed contingently true information in an assertional network compared to definitional networks. In addition, assertional networks are explicitly marked with a modal operator. The conceptual structures underlying natural language semantics are proposed as models.
3. If a network uses implication as the primary relation for connecting nodes, it is named as **implication network**. The representation patterns use beliefs, causality, or inferences for building the network.
4. Mechanisms like the transient or attached procedure used in **executable networks** that helps to create inferences, pass messages, or search for patterns.
5. **Learning networks** acquire knowledge from the real-time examples. Addition or deletion of arcs or nodes from previous network gives new knowledge. If this knowledge is using numeric values as its weight then it can exchange the weight associated with each arc or node.
6. A combination of two or more techniques can incorporate to form a new architecture called **hybrid networks**. This network is a single or a separate network that needs to be a closely interacting network.

Examples of semantic networks are given in Figure 6.14 and Figure 6.15. By traversing network (Figure 6.14) it is found that, Hari has a head (by inheritance) and that certain concepts are related in some ways (For example, apples and elephants). It is not well-defined the meaning of semantic networks, always. Some times confusions arises while asking questions from the semantic networks. A set of questions such as:

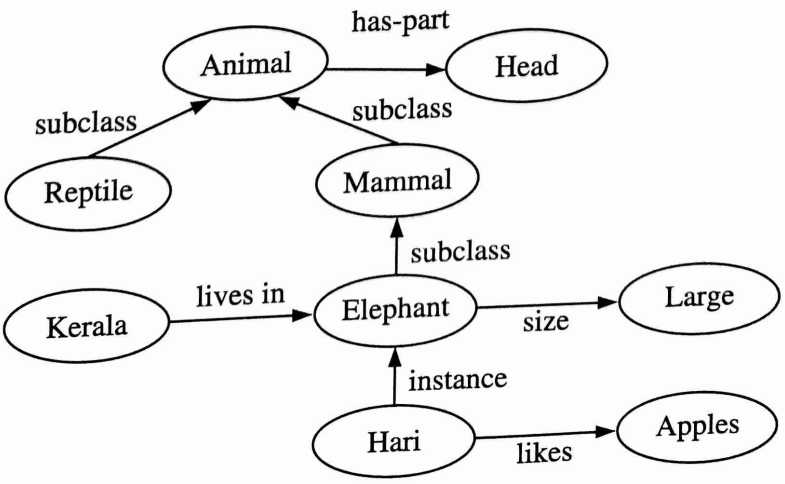

Figure 6.14 A semantic network—animals.

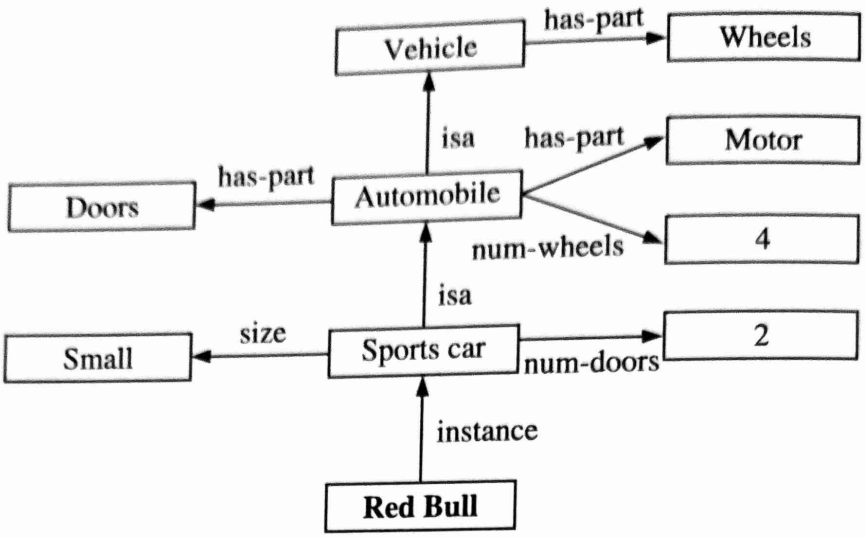

Figure 6.15 A semantic network—vehicle.

Are all elephants big, or just typical elephants?

Do all elephants live in the same Kerala?

Do all animals have the same legs?

These things are defined by machine processing. Modern trend is to have particular knowledge representation languages that look a bit like frames to users. However, which use logic to define a relation mean and do not provide the full power of predicate logic, but a subset that allows efficient inference.

There are some problems with semantic networks like they are computationally costly at run-time as we need to traverse the network to answer some question. We need to traverse the entire network in the worst case, because it is a graph structure. The search may be failed if the knowledge is not existing in the network node. Since the model is like human associative memory, i.e. stores information using associations. It is not practical to build a large semantic network because that is logically inadequate.

6.7 Exercises

1. What do you mean by knowledge representation?
2. Explain the concept of symbolic representation of knowledge.
3. State and prove Unification algorithm.
4. Convert the following facts to First order logic:
   "Everyone is related to someone who is retired."
   "For any x and y, if x is older than y, then y is younger than x".
   "All freighters are ships".
   "The mother of the child is the female parent".
   "Some intelligent students study computer science"
5. Consider the problem of deciding which clothes to wear using knowledge such as:
   "Wear casual clothes unless they are not clean or important meeting occurs today".