CHAPTER 6

KNOWLEDGE REPRESENTATION [+]

6.1. **Introduction**

In designing an expert system, possibly, the biggest problem is abstracting the knowledge of domain experts and putting it into an objective form for the knowledge base. The proper selection and design of a suitable knowledge representation scheme should be in tune with the requirements of the application domain. In addition, the proper selection should also depend on certain important properties of a scheme like expressive power and adequacy in context to the application domain. In this chapter, we have tried to analyse some of these issues from the viewpoint of an expert system designer.

Section 6.2 will be devoted to describe some knowledge representation schemes from the literature. In section 6.3, we shall analyse the relative suitability of such schemes as described in section 6.2. In section 6.4, we have presented some representative expert systems and ES-developmental tools along with the KR-schemes and control mechanism, they use. In section 6.5, the knowledge of the present problem domain has been represented in different schemes as discussed in section 6.2. Finally, we end up with some discussions.

6.2. **Some knowledge representation (KR) schemes**

Knowledge is as much an essential ingredient to the artificial intelligence of a computer as it is also to the natural intelligence of a person. A knowledge representation method is the way a knowledge engineering models the facts and relationships of the domain knowledge. The two types of knowledge that need to be represented in a computer are declarative knowledge and procedural knowledge. Declarative knowledge signifies facts about objects, events, and about how they relate to each other and procedural knowledge signifies the way to use the declarative knowledge.

In the present state-of-the-art of the subject there is no best theory of knowledge representation and so there is no best way to represent knowledge. Each method has its own advantages and disadvantages.

[+] This is based on the publication [Modelling, Measurement and Control, C, AMSE Press, vol. 31, no. 2, 1992, 13-24] of the author.

Several common knowledge representation schemes have been discussed so far in the literature [1-6] including logic, semantic networks, frames, OAV-triplets, scripts and production rule systems as classical methods; and the relatively new paradigm under development : object-oriented (O-O) approach.

6.2.1. **Logic as knowledge representation in expert system**

The formal logic systems used to represent declarative knowledge in AI are propositional calculus, predicate calculus, and first order predicate calculus.

An inference in propositional calculas is as follows :

- Pushy is a cat.
- If Pushy is a cat then she is a mammal.

In such a case according to an inference rule in propositional calculus known as **modus-ponens** the following must be true :

- Pushy is a mammal.

A generalisation like following is possible in predicate calculus.

- Pushy is a cat.
- All cats are bigger than all mice.

If the above two statements are true then

- Pushy is bigger than all mice.

First-order predicate calculus is created by adding functions and some other analytical features. As for example, a function "is-owned-by" is represented as

(is-owned-by (Pushy Peter)).

It may be mentioned that, the very first AI program "The Logic Theorist" was written using formal logic. A somewhat less formal systems of logic, such as fuzzy logic are used to reprsent concepts that are relatively vague and approximate such as "moderately expensive", "somewhat tall", "not so beautiful" etc.

There are advantage and disadvantage of using logic. The idea of logic, having matured for centuries are understood by intellectual community throughout the world. But, everybody will agree that, only a limited portion of intelligent human behaviour can be described in terms of logic [7]. However, different aspects of logic, along with the proofs and rules of inferences such as **modus-ponens**, **modus tolens, hypothetical syllogism** and **resolution** are treated in different text books on AI and Expert Systems.

6.2.2. Knowledge representation using semantic nets

The term semantic net is used to describe a knowledge representation method based on a network structure. Semantic nets were originally developed for use as psychological models of human memory but are now a standard representation method for AI and expert systems. A semantic network method represents knowledge using two tuples : (N, L). N is a set of nodes representing objects and descriptors. An object may be a physical or conceptual entity. A descriptor provides additional information about an object. L is a set of links connecting the nodes and representing the relations among them. Mainly three types of links are used : has-a link, is-a link, and definitional links. A has-a link shows that a node has a certain property. An is-a link represents class and instance relationships. A definitional link is used for representing declarative relations. Fig. 6.1 shows the structure of a semantic net.

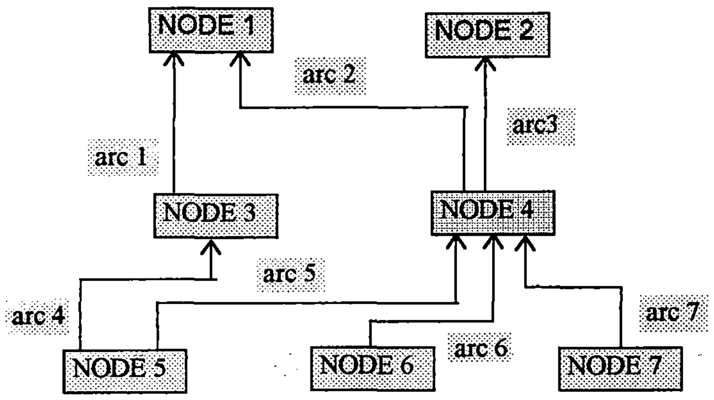

Fig. 6.1 Structure of semantic net

For example, a simple description of a bluebird might be "a bluebird is a small blue-colored bird and a bird is a feathered flying vertebrate". This may be represented as a set of logical predicates [8] :

size(bluebird, small).
hascovering(bird,feathers).
hascolour(bluebird, blue).
hasproperty(bird, flies).
isa(bluebird, bird).
isa(bird, vertebrate).

This description could also be represented graphically by using the links in a graph instead of predicates to indicate relations (fig.6.2). This description, called a semantic network, is a fundamental technique for representing semantic meaning. Because relationships are explicity denoted by the links of the graph, an algorithm for reasoning about the domain could make relevant associations simply by following links. In the bluebird illustration, a system need only follow two links in order to determine that a bluebird is vertebrate. This is more efficient than exhaustively searching a data base of predicate calculus descriptions of the form isa (X,Y).

In addition, knowledge may be organized to reflect the natural class-instance structure of the domain. Certain links in a semantic network (the ISA links in fig.6.2) indicate class membership and allow properties attached to a class description to be inherited by all members of the class. This inheritance mechanism is built into the language itself and allows knowledge to be stored at the highest possible level of abstraction. Inheritance is a natural tool for representing taxonomically structured information and ensures that all members of a class share common properties.

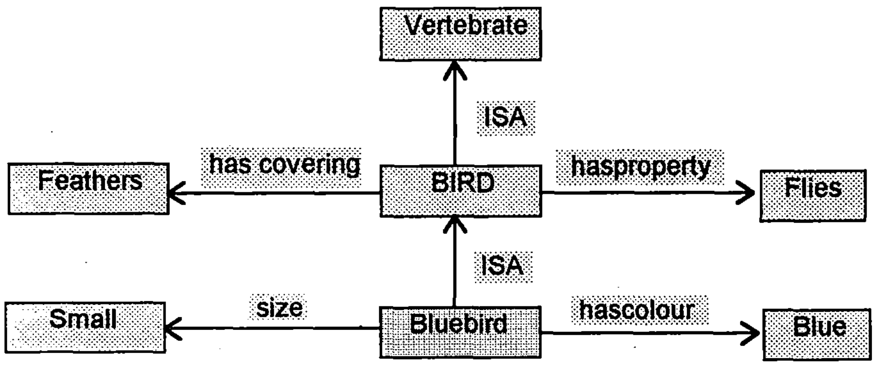

Fig. 6.2 Semantic network description of a bluebird

6.2.3. **Knowledge representation using rules**

A rule has two parts. The first part is a premise of conditions connected by logical-AND or logical-OR relationships. The second part is a conclusion. When the premise of a rule is true, the conclusion of the rule will become true. In some systems rules may be implemented by semantic networks or OAV, as in MYCIN [9], the medical diagnostic system developed at Stanford University. Alternatively, rules may be represented by frames, as in IntelliCorp's knowledge engineering environment [10].

In expert systems jargon the term rule has a much narrower meaning than it does in ordinary language. It refers to the most popular type of knowledge representation technique, the rule-based representation. Rules provide a formal way of representing recommendations, directives, or strategies, they are often appropriate when the domain knowledge results from empirical associations developed through years of experience solving problems in an area. Rules are expressed as IF-THEN statements, as shown below [11] :

(1) If a flammable liquid was spilled,
    called the fire department.

(2) If the pH of the spill is less than 6,
    the spill material is an acid.

(3) If the spill material is an acid,
    and the spill smells like vinegar,
    the spill material is acetic acid.

These are rules that might exist in a crisis management expert system for containing oil and chemical spills. Rules are sometimes written with arrows ($\rightarrow$) to indicate the IF and THEN portions of the rules.

Rule 2 in this notation would look like :

```
[2] If the pH of the spill →  the spill material
        is less than 6              is an acid.
```

In a rule-based expert system, the domain knowledge is represented as sets of rules that are checked against a collection of facts or knowledge about the current situation. When the IF portion of a rule is satisfied by the facts, the action specified by the THEN portion is performed. When this happens the rule is said to fire or execute. A rule interpreter compares the IF portions of rules with facts and executes the rule whose IF portion matches the facts, as shown in fig. 6.3.

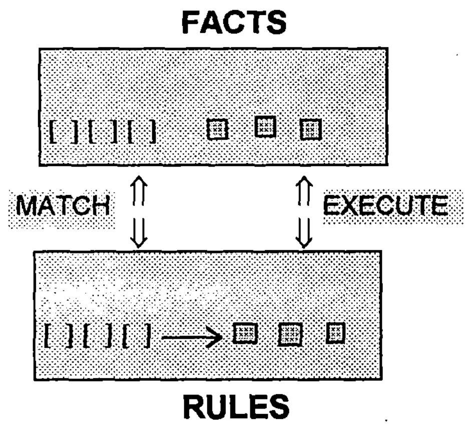

Fig. 6. 3 The rule interpreter cycles through a match-exucate sequence

The rule's action may modify the set of facts in the knowledge base, for example, by adding a new fact, as shown in fig. 6.4. The new facts added to the knowledge base can themselves be used to form matches with the IF portion of rules as illustrated in fig. 6.5. The action taken when the rule fires may directly affect the real world, as shown in fig. 6.6.

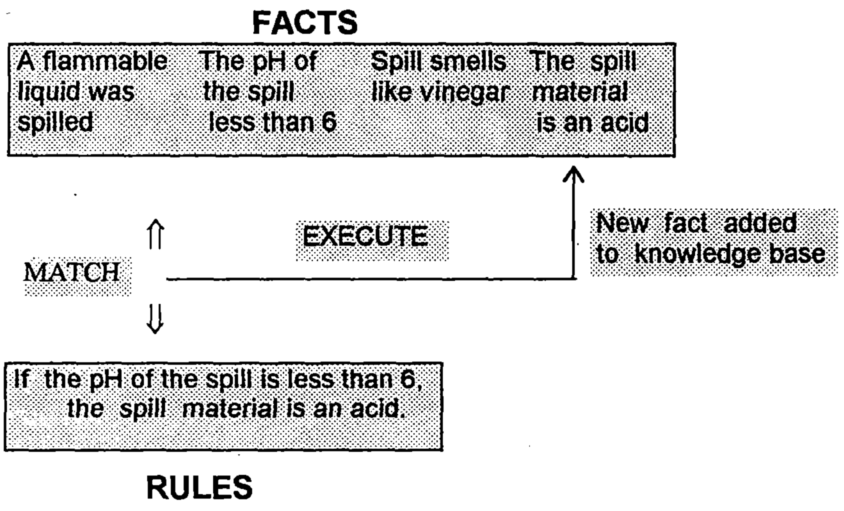
FACTS
A flammable liquid was spilled
The pH of the spill less than 6
Spill smells like vinegar
The spill material is an acid
EXECUTE
New fact added to knowledge base
MATCH
If the pH of the spill is less than 6, the spill material is an acid.
RULES

Fig. 6.4 Rule execution can modify the facts in the knowledge base

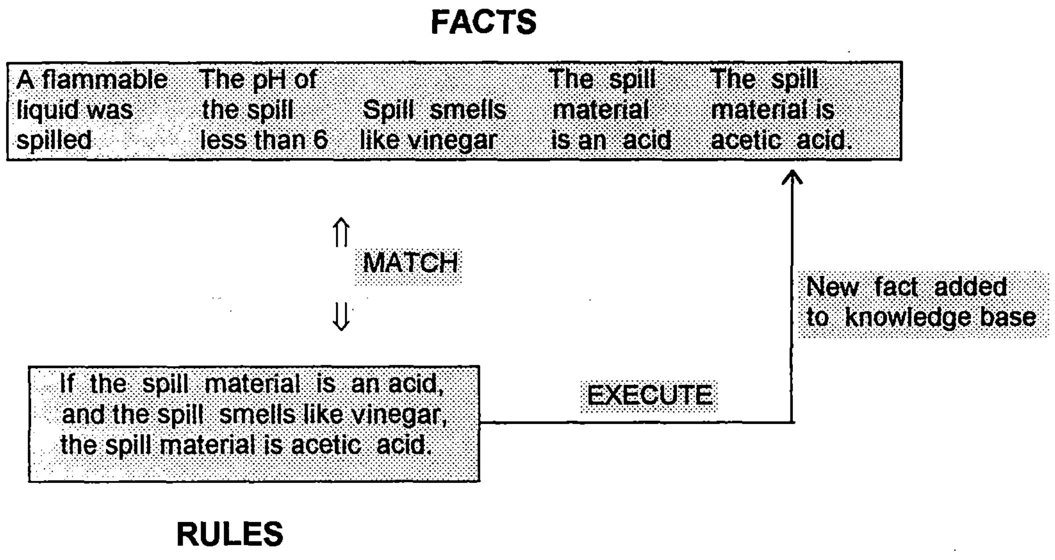
FACTS
A flammable liquid was spilled
The pH of the spill less than 6
Spill smells like vinegar
The spill material is an acid
The spill material is acetic acid.
MATCH
New fact added to knowledge base
EXECUTE
If the spill material is an acid, and the spill smells like vinegar, the spill material is acetic acid.
RULES

Fig. 6.5 Facts added by rules can match rules

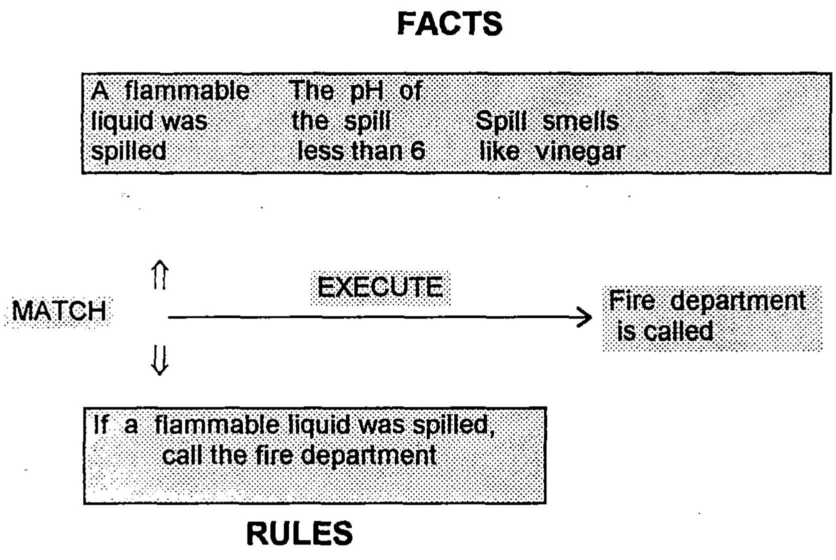
FACTS
A flammable liquid was spilled
The pH of the spill less than 6
Spill smells like vinegar
MATCH
EXECUTE
Fire department is called
If a flammable liquid was spilled, call the fire department
RULES

Fig. 6.6 Rule execution can affect the real world

This matching of rule IF portions to the facts can produce what are called inference chains. The inference chain formed from successive execution of rules 2 and 3 is shown in fig. 6.7. This inference chain indicates how the system used the rules to infer the identify of the spill material. An expert system's inference chains can be displayed to the user to help explain how the system reached its conclusions.

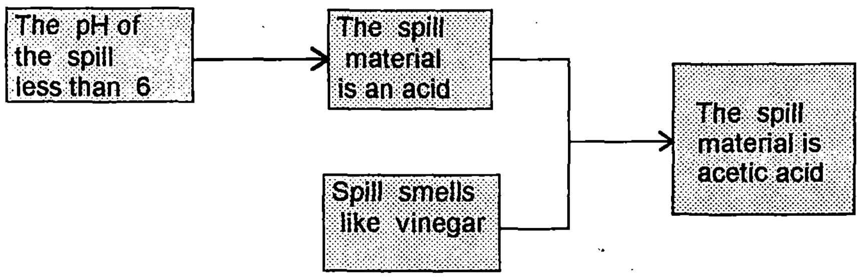
The pH of the spill less than 6
The spill material is an acid
Spill smells like vinegar
The spill material is acetic acid

Fig. 6.7 Inference chain for inferring the spill material

There are two important ways in which rules can be used in a rule-based system, one is called forward chaining (data driven) and the other backward chaining (goal driven). The spill material example just presented used forward chaining. Fig. 6.8 shows in more detail how forward chaining works for a simple set of rules.

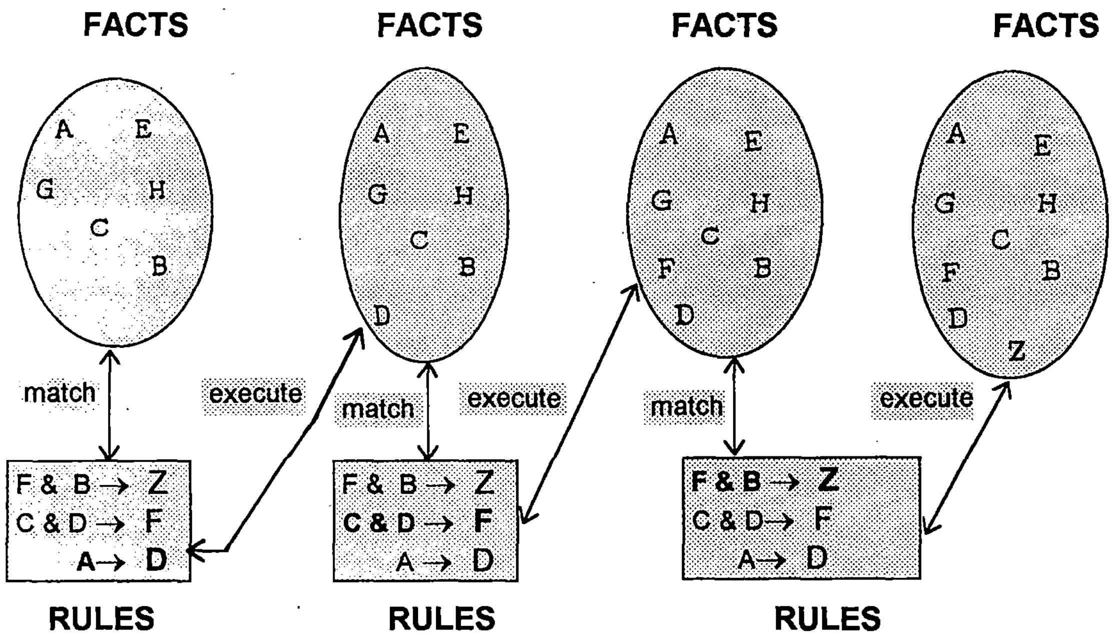

Fig. 6.8 An example of forward chaining

The rules in this example use letters to stand for situations or concepts.

F and B $\longrightarrow$ Z

means

IF : both situation F and situation B exist

THEN : situation Z also exists.

The known set of facts we will call the data base.

Let's consider how these rules work. We'll assume that each time the set of rules is tested against the data base, only the first (topmost) rule that matches is executed. That's why, in fig. 6.8 the rule A $\rightarrow$ D is only executed once even though it always matches the data base.

The first rule that fires is A $\rightarrow$ D because A is already in the data base. As a consequence of that rule, the existence of D is inferred and D is placed in the data base. That causes the second rule C & D $\rightarrow$ F to fire, and as a consequence F is inferred and placed in the data base. This in turn causes the third rule F & B $\rightarrow$ Z to fire, placing Z in the data base.

This technique is called forward chaining because the search for new information seems to be proceeding in the direction of the arrows separating the left-hand and right-hand sides of the rules. The system uses information on the left-hand side to derive information on the right. The inference chain produced by the example in fig. 6.8 is shown in fig. 6.9. Situation Z was inferred to exist as well as situations F and D.

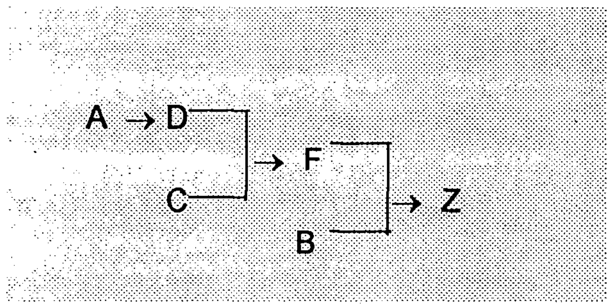

Fig. 6.9 Inference chain by the example in fig. 6.8

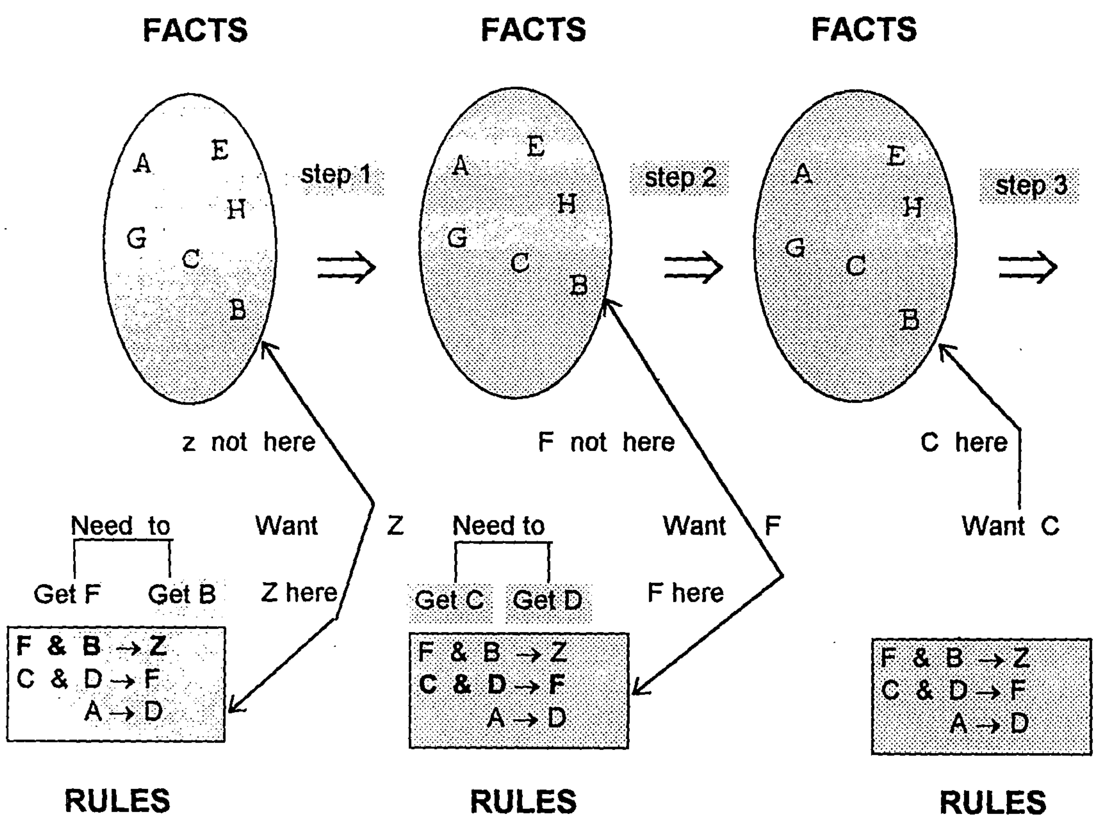

Fig. 6. 10 An example of backward chaining

A real expert system wouldn't have just three rules, it would have hundreds or even thousands of them. If you used a system that large just to find out about Z, many rules would be executed that had nothing to do with Z. A large number of inference chains and situations could be derived that were valid but unrelated to Z. So if your goal is to infer one particular fact, like Z, forward chaining could waste both time and money.

In such a situation backward chaining might be more cost-effective. With this inference method the system starts with what it wants to prove, e.g., that situation Z exists, and only executes rules that are relevant to establishing it. Fig. 6.10 shows how backward chaining would work using the rules from the forward chaining example.

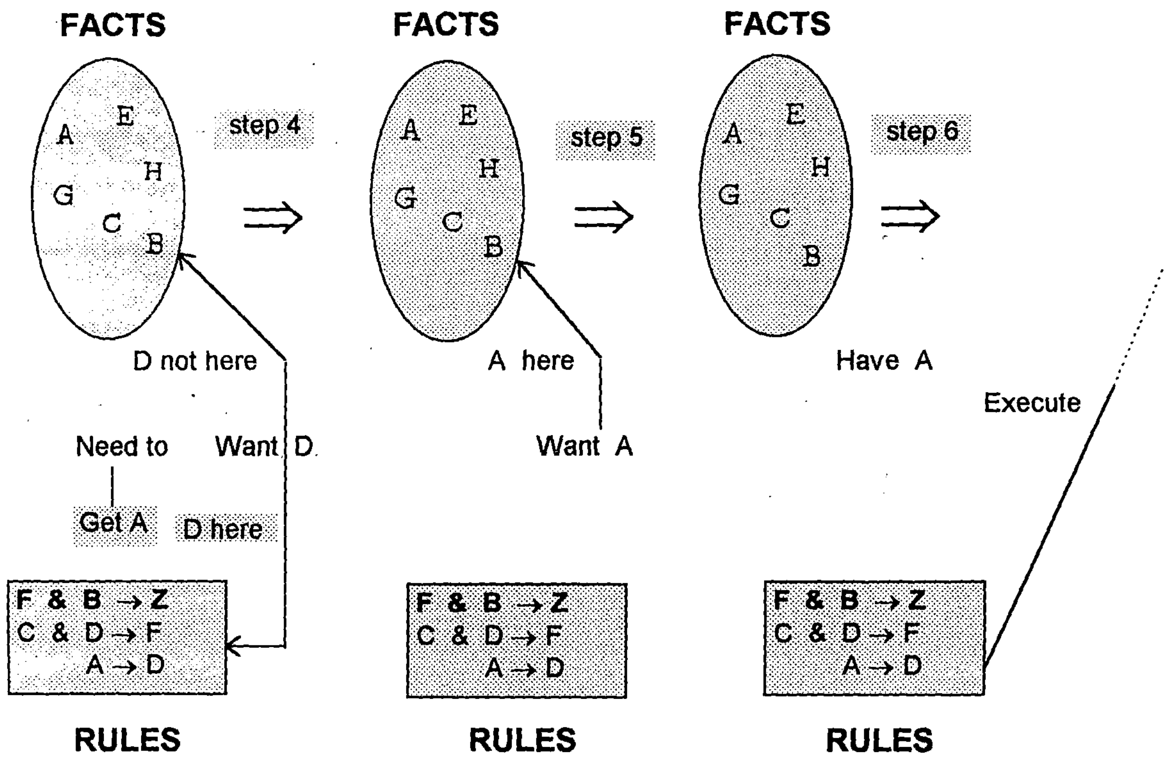

Fig. 6. 11 Continuation of fig. 6.10

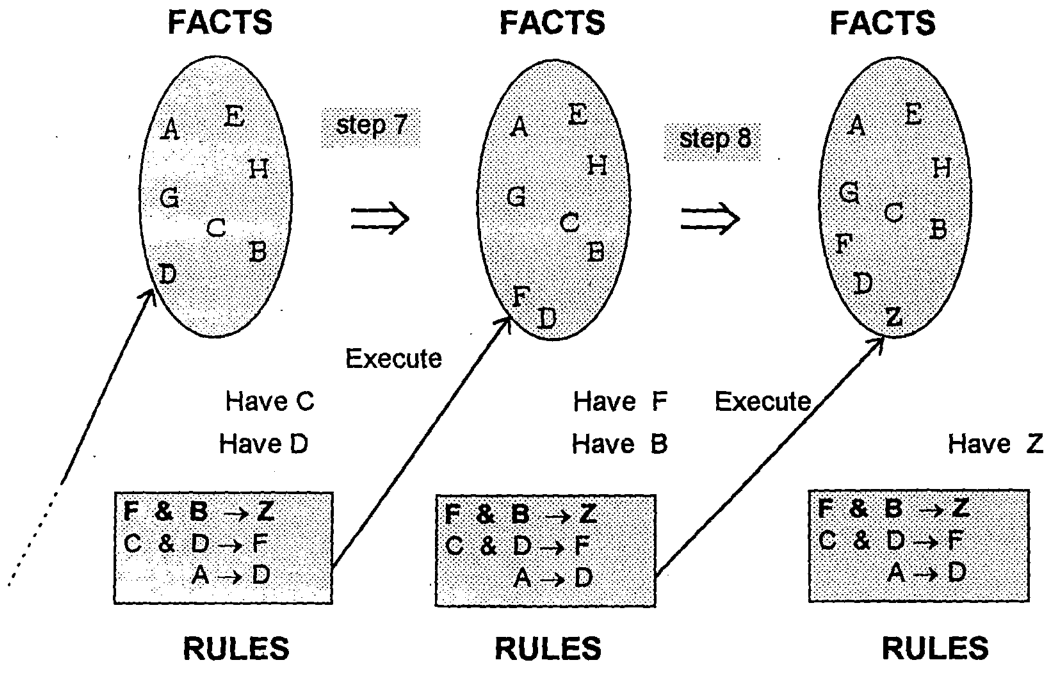

Fig. 6.12 Continuation of fig. 6.11

In step 1 the system is told to establish (if it can) that situation Z exists. It first checks the data base for Z, and when that fails, searches for rules that conclude Z, i.e., have Z on the right side of the arrow. It finds the rule F and B $\rightarrow$ Z, and decides that it must establish F and B in order to conclude Z.

In step 2 the system tries to establish F, first checking the data base and then finding a rule that concludes F. From this rule, C and D $\rightarrow$ F, the system decides it must establish C and D to conclude F.

In steps 3 through 5 the system finds C in the data base but decides it must establish A before it can conclude D. It then finds A in the data base.

In steps 6 through 8 the system executes the third rule to establish D, then executes the second rule to establish F, and finally executes the first rule to establish the original goal. The inference chain created here is identical to the one created by forward chaining. The difference in the two approaches hinges on the method in which data and rules are searched.

Depth-first search and Breadth-first search [4]

In addition to specifying a search direction (data-driven or goal-driven), a search algorithm determines the order in which states are examined in the tree or graph. This section considers two possibilities : depth-first and breadth-first search.

Depth-first search

A depth-first search begins at the root node and works downward to successively deeper levels. An operator is applied to the node to generate the next deeper node in sequence. This process continues until a solution is found or backtracking is forced by reaching a dead end.

A simple depth-first search is illustrated in fig. 6.13. The numbers inside the nodes designate the sequence of nodes generated or searched. This process seeks the deepest possible nodes. If a goal state is not reached in this way, the search process backtracks to the next highest level node where additional paths are available to follow. This process continues downward and in a left-to-right direction until the state goal is discovered. Here, the search would actually end at node 13.

When a dead-end node is discovered, such as node 4 in fig. 6.13, the search process backtracks so that any additional branching alternative at the next higher node level is attempted. The search backs up to node 3. It has no alternate paths, so the search backtracks to node 2. Here, another path through node 5 is available. The path through node 6 is explored until its depth is exhausted. The backtracking continues until the goal is reached.

The depth-first serch guarantees a solution, but the search may be a long one. Many different branches will have to be considered to a maximum depth before a solution is reached. (By setting a "depth bound", it is frequently possible to reduce the search.) The method is especially attractive in case where short paths exist and where there are no lengthy sub-branches.

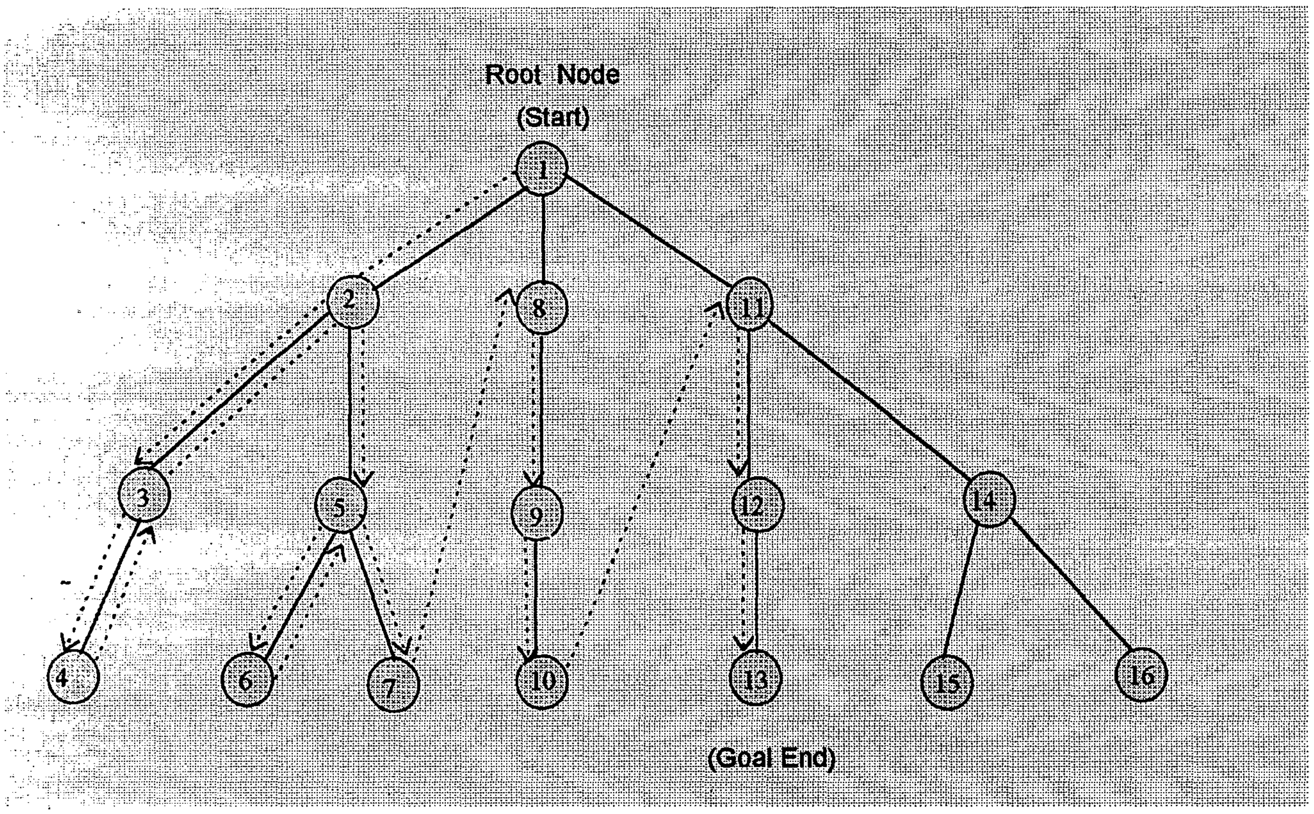

Fig. 6.13 Depth-first search

Breadth-first search

A breadth-first search examines all of the nodes (states in a search tree, begining with the root node. The nodes in each level are examined completely before moving on to the next level. A simple breadth-first search is illustrated in fig. 6.14. The numbers inside the node circles designate the sequence in which the nodes are examined. In this instance, the search (follow the broken line) would actually end at node 7, as that is the goal state.

A breadth-first search of the state space will usually find the shortest path between the initial state and the goal state, with the least number of steps.

The process usually starts at the initial state node and works downward in the tree from left to right. A terminal node is not necessarily a goal node; it can be a dead-end node. Breadth-first procedures are good when the number of paths emanating from each goal is relatively small and where the number of levels in each branch is of a different depth (number of levels).

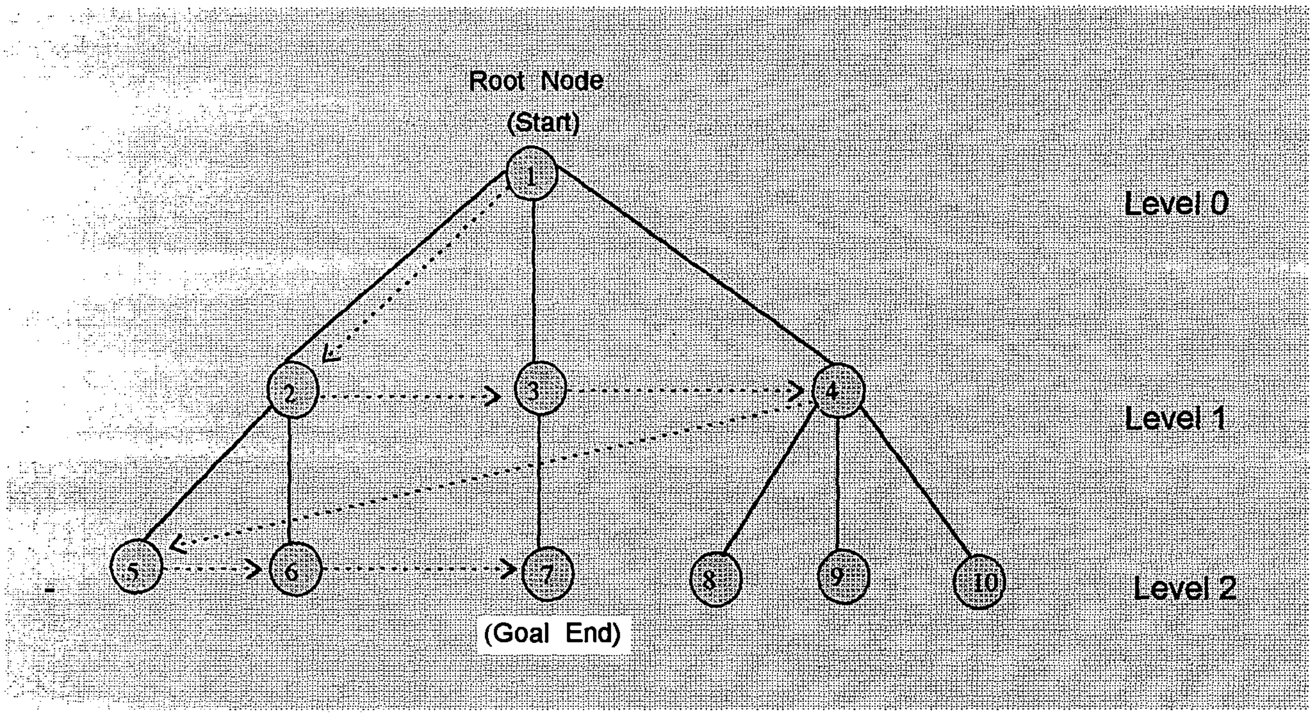

Fig. 6.14 Breadth-first search

6.2.4. **Knowledge representation using frame concepts**

A frame is used to describe an object [12]. It is composed of slots storing information associated with the object. The function of the slots is similar to that of the attributes in OAV. However, frames differ from OAV in that the slots may contain default values, pointers to other frames, sets of rules, or procedures. Frames may also be linked to allow for inheritance. So, frames and OAV can be considered special cases, or subsets, of semantic networks. The representational power of the three systems is the same. The difference lies in the structure and concept of their knowledge organizations.

According to Marvin Minsky [12] when we mentally recall the image of a particular object, we recall a group of typical attributes of that object at the same time - this cohesive grouping of attributes is called Frames. For example, if somebody mentions about a chair, that would "trigger" a series of expectations - such that it is an object with four legs, a seat, a back, and possibly but not necessarily two arms.

A preconception about the colour may not be there but a general expectation of size will be there [3, 5]. A frame of the word " chair" is shown in figure 6.15.

FRAME : CHAIR

PARTS	:	SEAT, BACK, LEGS, ARMS
NUMBER OF LEGS	:	4
NUMBER OF ARMS	:	0 OR 2
DEFAULT	:	0
COLOUR	:	ANY
SIZE (in feet)	:	
HEIGHT	:	2.5 - 5
WIDTH	:	1 - 3
DEPTH	:	1 - 3
STYLES	:	ROCKING, RECLINING, OFFICE

Fig. 6.15 A frame for the word "CHAIR"

6.2.5. **Knowledge representation using scripts**

Another knowledge representation system that is especially useful in the area of natural language understanding is a system called scripts, proposed by Roger Schank [13] at Yale University. Scripts are composed of a series of slots that describe, in sequence, the events that we expect to take place in familiar situations. Just as the concept of frames is based on the assumption that we have a set of expectations about objects, the use of scripts assumes that we also expect certain sequences of events to occur in particular times and places. Schank and Childers in their book [13] the Cognitive Computer uses a resultant script. Visit to a restaurant is composed of a series of scenes, for example an entering scene, a sitting scene, an ordering scene, an eating scene, a bill payment scene etc. Fig. 6.16 shows a script for entering scene.

```
SCRIPT          : RESTAURANT
SCENE           : ENTERING
                  GO INTO THE RESTAURANT
                  LOOK AT THE TABLES
                  DECIDE WHERE TO SIT
                  GO TO TABLE
                  SIT.
```

Fig. 6.16 A script for entering a restaurant

6.2.6. **Object-attribute-value triplets as KR scheme**

The object-attribute-value triplets (OAV-triplets) method represent knowledge using three tuples : (O, A, V). O is the set of objects, which may be physical or conceptual entities. A represents the set of attributes characterising the general properties associated with objects. V (values) specify the nature of the attributes.

The OAV method is actually a special form of semantic network. The relation between an object and an attribute is a has-a link, and the relation between an attribute and a value is an is-a link. The objects, attributes, and values of OAV are equivalent to the nodes in semantic networks. Knowledge can be divided into dynamic and static portions. The triplet values are the dynamic portion. These values may change, but the static portion (usually facts and rules) remains unchanged for different consultations.

An expert system stores data about real-world entities. In knowledge representation theory, the real-world entities are objects. Each object has one or more attributes or properties, and the attributes have values; for example,

Object	Attribute	Value
Bus	Colour	White
John	Fever	High

The object is **bus**, the attribute is **colour**, and the value is **white**. Another example, the object is **John**, the attribute is **fever**, and the value is **high**.

OAV is more structured than a semantic network. However, when the number of objects increases, an OAV system becomes difficult to manage.

6.2.7. **Object-Oriented approach**

The world consists of different objects. An object is an independent entity represented by some data and a set of operations (methods and capabilities) [14]. Therefore, an object can be used to represent a variety of knowledge. Knowledge (K) can be formally represented by three tuples, K = (C, I, A). C is a set of classes represented by class objects. I is a set of instances represented by instance objects. A is a set of attributes possessed by the classes and instances.

6.2.7.1. **Classes**

A class is a description of a group of similar instance objects [14]. It is a mold that determines the behavior of its instances. Each class has a unique name and a set of attributes that define the properties of the class. A class may be a sub-class of another class and may inherit properties from its parent class as discussed in **sub-section** 6.2.7.4.

Five sets of attributes may exist in a class object :

(a) Name - the name of the class; used to reference a class in the system.

(b) Super_class - the name of the parent of the class.

(c) Class_variables - a set of variables shared among all instances of the class, that is, global variables accessible by all instances of the class. Get and Store operations may be performed on them.

(d) Instance variables - the set of variables possessed by each instance of the class. Variables may be classified into data and database data. Each instance may have different values for these variables.

(e) Class attributes - the set of methods, external methods and rules shared by all instances of the class.

6.2.7.2. **Instance objects**

Instance objects are members of classes. Their properties are defined by their parent classes. Each instance object consists of three sets of attributes :

(a) Name - the name of the object, which is unique in the system. It is used to identify the object.

(b) Class - the class that contains the object.

(c) Instance attributes - attributes belonging to the instance object. Some operations may be performed on these attributes. The behavior of the object is determined by the values of these attributes.

In general, an instance object i is defined by its class and the values of its attributes : $i \in I$, where I is the set of instances in the system; $\forall i \in I$, attribute x with any value v is valid for i, if val (x) = v and x are defined in class (i), where class (i) is a function that returns the name of the class of instance i and val (x) is a function that returns the value for attribute x.

6.2.7.3. **Attributes** and **methods / operations**

The property of an attribute is determined by its type and value. The type of an attribute is defined by its class, while the value may be defined in its class or its instance object. A set of generic attributes can be associated with the every object in a class, say furniture, for example. All furnitures has a cost, dimensions, weight, location, and color among many possible attributes.

Methods are a kind of attribute belonging to objects. They are used to represent capabilities, not to store data, and are defined in classes. Methods cannot be modified during consultations.

6.2.7.4. **Inheritance**

Properties of a class can be inherited from its parent's class. This feature permits factoring knowledge into a class hierarchy. Thus, it encourages modular design of knowledge. The system adopts the inheritance rules, which are similar to those in smalltalk [14], as follows :

(a) If class A inherits from class B, then the objects of class A support all operations supported by objects of class B.

(b) If class A inherits from class B, then class A's attributes are a superset of class B's attributes.

Therefore, $\forall c \in C$ (the set of all classes), attribute x is valid for c if either $x \in$ class_ attributes (c), where class_attributes (c) = the set of attributes of c.

For example, Chair is a member of the class furniture. Chair inherits all attributes defined for the class. This concept is illustrated schematically in fig. 6.17.

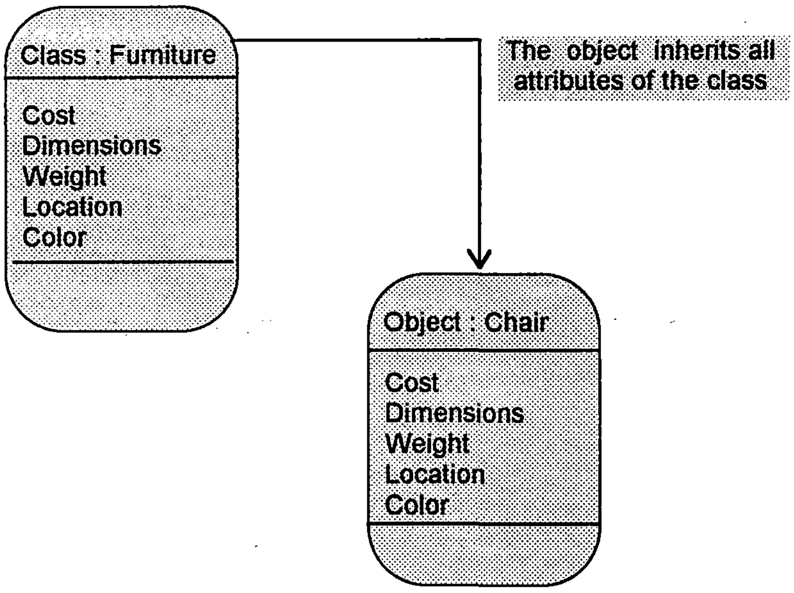

Fig. 6.17 Inheritance from class to object

Once the class has been defined, the attributes can be reused when new instances of the class are created. For example, assume that we were to define a new object called **chable** (a cross between a chair and a table) that is a member of the class **furniture**. **Chable** inherits all of the attributes of **furniture** [15].

Every object in the class furniture can be manipulated in a variety of ways. It can be bought and sold, physically modified, or moved from one place to another. Each of these operations will modify one or more attributes of the object. For example, if the attribute **location** is actually a composite data item defined as :

Location = building + floor + room

Then an operation named **move** would modify one or more of the data items (**building**, **floor**, or **room**) that comprise the attribute **location**. To do this, move must have "knowledge" of these data items. The operation **move** could be used for a chair or a table, as long as both are instances of the class **furniture**. All valid operations (e.g. **buy**, **sell**, **weigh**) for the class **furniture** are "connected" to the object definition as shown in fig. 6.18 and are inherited by all instances of the class.

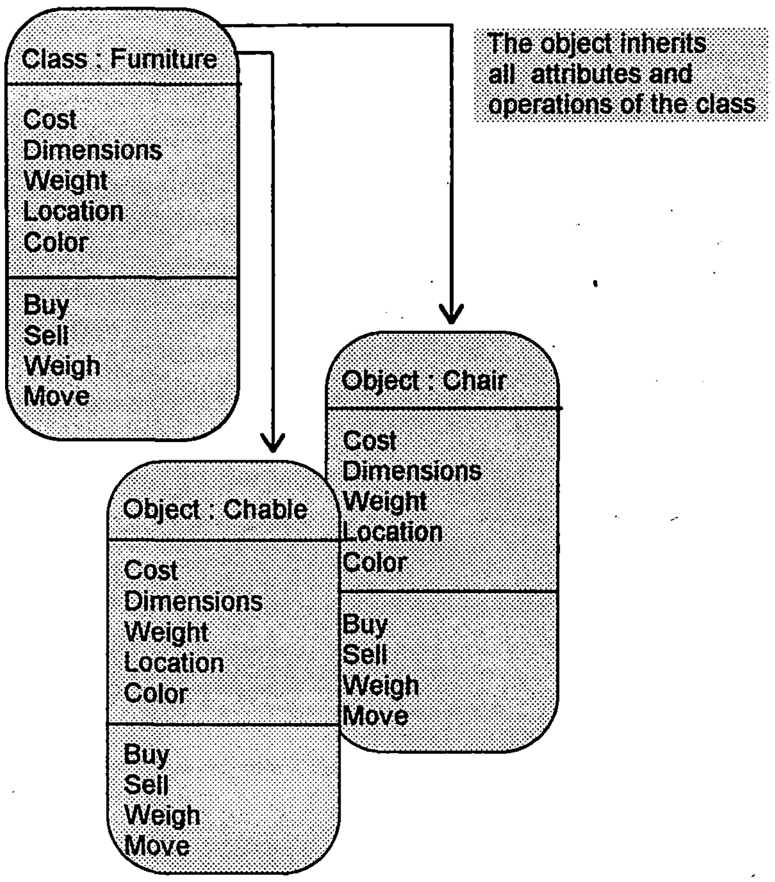

Fig. 6.18 Inheritance of operations from class to object

-The object **chair** (and all objects in general) encapsulates data (the attribute values that define the chair), operations (the actions that are applied to change the attributes of chair), other objects (composite objects can be defined [16]), constants (set values), and other related information. Encapsulation means that all of this information is packaged under one name and can be reused as one specification or program component.

6.2.7.5. **AI, Expert systems** and **O-O technology**

The object-oriented (O-O) paradigm does have some contributions to AI and expert systems technology. In recent years, the term or adjective 'object-oriented' has become a popular slogan of any kind of systems regardless of its actual qualities. But however, we must admit that behind the slogan there must be some interesting ideas and concepts people suggest for developing large, integrated systems. Although there is the lack of standard definition of what is the O-O approach, but however, the properties like encapsulation, inheritance, polymorphism are considered useful for O-O approach. O-O is now being exploited for analysis and design, and people are sharing their experiences.

The ideas behind object oriented programming (OOP) and object oriented technology (OOT) date back to the forties [17]. These ideas, however, were not put into practice until the introduction of the Simula_67 programming language [18]. Simula, a superset of Algol, was designed for describing a wide class of discrete event simulations and implementing them for simulations. Simula objects represent data and an operation on the data. These objects communicate with each other through messages to determine their next action. Although primitive by today's standards, Simula provided the first insight into the value of OOP.

The form of OOP we are accustomed to seeing took shape in the seventies with the development of Smalltalk at the Xerox Palo Alto Research Centre. Although Smalltalk is used to develop expert systems, its real value is that it offers a user-friendly programming environment.

What made Smalltalk easy to use and conceptually appealing was the extensive use of techniques commonly found today in OOP languages : class / object representations, inheritance, message-passing and encapsulation, to name a few. Researchers at Xerox Palo Alto Research Centre found that these techniques enabled a programmer to easily perceive an object system's structure and operation and to use this understanding to efficiently develop an interface, or for that matter, an entire functioning program. OOP's intuitive approach was the key to Smalltalk's success. Programming solutions frequently followed the methods that humans use to address everyday problems.

Given Smalltalk's intuitive programming environment, coupled with AI researchers' interests in computers representing and reasoning with knowledge similarly to humans, it was only natural for these researchers to adopt object-oriented techniques. This trend was most noticeable during the eighties.

One of the most important events during the eighties that spurred the interest in AI was the marketing of expert system development shells. Most of the early shells were rule-based. However, given the appeal of object-oriented systems, as demonstrated by Smalltalk's success, the demand pushed vendors to offer tools with object-oriented techniques. These tools, commonly called frame-based development programs (but sometimes called hybrid tools), usually combine object-oriented techniques with rule-based programming. New procedural languages with object-oriented techniques also surfaced, such as Objective C, C++, Pascal Object, Modula-2 and Lisp extensions such as Scoops, Flavors, Loops and the Common Lisp Object System(CLOS).

Armed with powerful object-oriented shells and languages, expert system developers took aim at problems that were often out of the reach of rule-based approaches. A review of systems developed during the later eighties and early nineties clearly shows a swing toward object-oriented techniques [19]. This trend was due partly to the availability of relatively inexpensive frame-based shells that ran on a variety of platforms. Two of the earliest frame-based shells, the knowledge engineering environment from IntelliCorp and the automated reasoning tool from Inference, offered AI researchers powerful tools, but were costly and ran on mainframes or workstations, preventing their widespread use. In the mid-eighties, vendors began marketing cheaper object tools, many of which ran on a PC. This situation led to the accelerated development of frame-based expert systems. Most important, it open the door at most universities for teaching object-oriented technology (OOT) techniques to the next generation of AI researchers. Flourishing development of object-oriented knowledge-based systems continues. Vigorous development of object-oriented knowledge-based systems continues. Most corporations - including many in the Fortune 500 - are focusing on client-server and object-oriented problems. These organizations have come to recognize AI in general and OOT in specific, as a standard way of doing business. Whereas many of these companies first ventured into AI by forming a dedicated group of AI specialists, most of these specialists now work in the more traditional programming departments, where they routinely carry on their trade of knowledge-based programming.

A look at the recent marketing approach of vendors of AI object-oriented tools is also revealing. As any good business would do, these vendors have kept a finger in the air to sense the direction of their clients' interests. They found that although terms such as "AI" and "expert systems" might have fallen out of favour in some circles, their clients still wanted the object-oriented capability of their products. To go with the flow, these vendors began to advertise their products as "intelligent applications tools". AI capability was still there, but the idea of AI faded into the background. This presents an interesting situation : companies using AI but not promoting it, and vendors marketing products with AI capabilities but not advertising it. Although abandoning the AI label, both have created a new infrastructure on which to build the knowledge-based technology that should flourish in the latter part of the nineties. The irony : even if the spotlight is no longer on AI, AI's contributions will continue to positively affect future information processing, only under other labels [20].

6.3. **Analysing relative suitability**

The major advantage of semantic networks is flexibility, since new nodes and links can be defined as required without restriction. This flexibility also exists in object-oriented (O-O) knowledge representations where, by storing the names of their objects as the attributes of an instance object, relations between instance objects can be established dynamically. These relations have the same power as links in semantic networks; in fact, this object-oriented (O-O) construct can be viewed as dynamic semantic network. The is-a links of semantic networks can be implemented in object-oriented (O-O) representations by relationships between classes and sub-classes or between classes and instances. Has-a links can be implemented by the relationships between classes and attributes. Therefore, object-oriented knowledge representation has the same power as a semantic network but is much more structured.

A common disadvantage in semantic networks, rules, and OAV representations is that they are not structured enough. A significant increase in the number of objects or rules makes the system difficult to manage. This is because the knowledge cannot be modularized and interactions among rules and objects become too complex. When the value of an object or an attribute is modified, it is difficult to pinpoint the effects on the whole system. Therefore, such knowledge representations are difficult to develop and maintain, especially for a large knowledge base. The encapsulation property and structuredness of object-orienred (O-O) knowledge representations give them a distinct edge over these three representations.

Frames are more structured than semantic networks, rules, and OAV representations, since related attributes and rules can be grouped into frames hierarchically. However, modularity of knowledge represented in frames cannot be clearly defined, and frame representation lacks flexibility. In a frame system, relationships between frames may be member or subclass links and thus are not unique. Moreover, in some systems, a rule is represented by a frame linked to another frame with a special relationship. These factors greatly reduce the structure in a frame system. In object-oriented (O-O) knowledge representation, which is quite similar to frames, knowledge can be arranged in a hierarchical form using classes. However, a subclass link is the only possible relationship between two classes, an is-a link is the only possible relationship between a class and an instance object, and rules are defined as methods in classes - clear-cut distinctions that reduce ambiguity and improve understandability.

In tune with our identified key requirements the domain of child growth and development (CGD) lays on an expert system, we now analyse the relative suitability of different KR schemes discussed in **section 6.2**. When the domain knowledge is vast and varied, the knowledge can become unmanageable. To handle a large knowledge base it is suggested [21] that the structuredness and modularity is necessary where knowledge is varied. A common disadvantage in Semantic Networks, Rules and OAV representations is that they are not structured enough [6]. It is very difficult to manage a system with these representations when the number of objects and rules increases significantly. According to some researchers [22], some applications such as engineering processes, manufacturing and communications are expected to contain 100,000 rules or more. It is then very difficult to pinpoint the effects on the whole system if a value of an object or an attribute is modified.

The major advantage with semantic networks is its flexibility in defining new nodes and links as when required. The type of flexibility is also with the O-O approach which may be viewed as a dynamic semantic networks. The O-O knowledge representation has the same power as of semantic networks but is much more structured. Frames are more structured than Semantic nets, rules and OAV representations, since related attributes and rules can be grouped into frames hierarchically. This is a passive data structure which lacks flexibility and the relationships within this system are not unique. The active data structure, the O-O representation of knowledge where declarative as well as procedural knowledge can be mixed, is structured and is much more meaningful semantically. The O-O form of KR encourages modular designs supporting the improvement of the efficiency of knowledge acquisition and management. The properties like encapsulation and inheritance of O-O approach are really attractive for large, integrated information systems. The encapsulation property prevents object manipulation except by defined operations. Inheritance is a valuable mechanism which enhances reusability and maintainability of software. Because this approach minimizes object interdependency [23] the knowledge can be structured.

A common disadvantage in OAV triplets and rules is that there may be some redundancy in information which may lead to some inconsistency. There is no such redundancy problem with Semantic Networks, frames and O-O forms. Moreover, O-O approach to KR supports high level of knowledge abstraction, an important advantage over other classical approaches. Considering all these factors, we advocate O-O representation to improve consistency, understandibility, maintainability and modifiability of knowledge base. Last, but not least, in the evolution of an expert system [11], prototyping may have an adverse impact on modifiability and maintainability of knowledge bases since these may be patched and modified several times. This may, however, be overcome by the use of O-O approach. As the system grows, the major changes will be with the addition of new objects or deletion of old objects rather than modifying the old objects. In this respect O-O approach is considered very useful for rapid prototyping, an added advantage.

6.4. **Some representative expert systems** and **ES-development tools**

The following table 6.1 represents some expert systems and ES-developmental tools [11, 24 - 26] with the kind of knowledge representation scheme(s) and control for knowledge base scanning.

Table 6.1

ES/ES-tools	Representation	Control
ADVISOR II	Rule-based	Backward chaining
EXSYS	Rule-based	Backward chaining
GURU	Rule-based	Backward chaining, Limited forward chaining
KES II	Rule-based, classes	Backward chaining, Limited forward chaining

LEONARDO	Rules, Frames, Procedures	Backward and forward chaining
XI PLUS	Rules, Induction	Control over direction
GOLDWORKS	Rules, Frames, Objects	Control over direction
NEXPERT	Rule-based	Forward, backward chaining
ESE	Rule-based	Backward chaining
OPS5	Rule-based	Forward chaining
EMYCIN	Rule-based	Restrictive backward chaining
LISP	Procedure-oriented, functional, symbolic expressions	Forward, backward chaining
PROLOG	Logic-based	Backward chaining
EXPERT	Rule-based	Forward chaining
ART	Rule-based, Frame-based	Forward and backward chaining
DUCK	Logic-based, Rule-based	Forward and backward chaining
GUSS/1	Rule-based	Backward and forward chaining
KES	Rule-based, Frame-based	Backward chainig
M.1	Rule-based	Backward chainig
OPS5	Rule-based	Forward chaining
RITA	Rule-based	Forward and backward chaining
SAVOIR	Rule-based	Backward and forward chaining
S.1	Rule-based, Frame-based	Backward chaining
ARBY	Rule-based	Backward chaining
Plant/cd	Rule-based	Backward chaining
XCON	Rule-based	Forward chaining
XSEL	Rule-based	Forward chaining
YES/MVS	Rule-based	Forward chaining
EL	Rule-based	Forward chaining
TALIB	Rule-based	Forward chaining
DELTA	Rule-based	Forward and backward chaining
SPERIL-I	Rule-based	Forward chaining
SPERIL-II	Rule-based	Forward and backward chaining

DIPMETER ADVISOR	Rule-based	Forward chaining
DRILLING ADVISOR	Rule-based	Backward chaining
MUD	Rule-based	Forward chaining
CODES	Rule-based	Backward chaining
FOLIO	Rule-based	Forward chaining
PROJCON	Rule-based	Backward chaining
DSCAS	Rule-based	Forward chaining
SAL	Rule-based	Forward chaining
TAXADVISOR	Rule-based	Backward chaining
IMACS	Rule-based	Forward chaining
PTRANS	Rule-based	Forward chaining
AI/RHEM	Rule-based	Forward chaining
BABY	Rule-based	Forward chaining
CLOT	Rule-based	Backward chaining
MEDICO	Rule-based	Forward chaining
MI	Rule-based	Forward chaining
MYCIN	Rule-based	Backward chaining
NEURAX	Rule-based	Forward and backward chaining
ONCOCIN	Rule-based	Forward and backward chaining
PUFF	Rule-based	Backward chaining
SPE	Rule-based	Forward chaining
THYROID MODEL	Rule-based	Forward chaining
WHEEZE	Frame-based	Backward and forward chaining
MES	Rule-based	Forward chaining
TATR	Rule-based	Forward chaining
PDS	Rule-based	Forward chaining
FAITH	Frame-based	Backward and forward chaining
LEVEL5 (OBJECT)	Large-hybrid-object-oriented, Rule-based	Forward and backward chaining

6.5. The present problem domain of child growth and development (CGD)

• OAV-triplets

The general form of OAV-triplets representation of the knowledge of **Appendix A** is shown as [27, 28] : (age-of-the-baby, activity, value).

For examples :

(1-month, Axial-muscle-tone, Head-drops),
(1-month, Axial-muscle-tone,Turns-head-from-side-to-side),
(1-month, Spontaneous-Gestures, Athetoid),
(1-month, Spontaneous-Gestures, Stays-lying-on-side),
(1-month, Rhythms-sleep, 21 hours),
(1-month, Rhythms-meals, 5 meals).

• Semantic networks

From the **Appendix A**, we show a small portion of the total semantic networks drawn as an example.

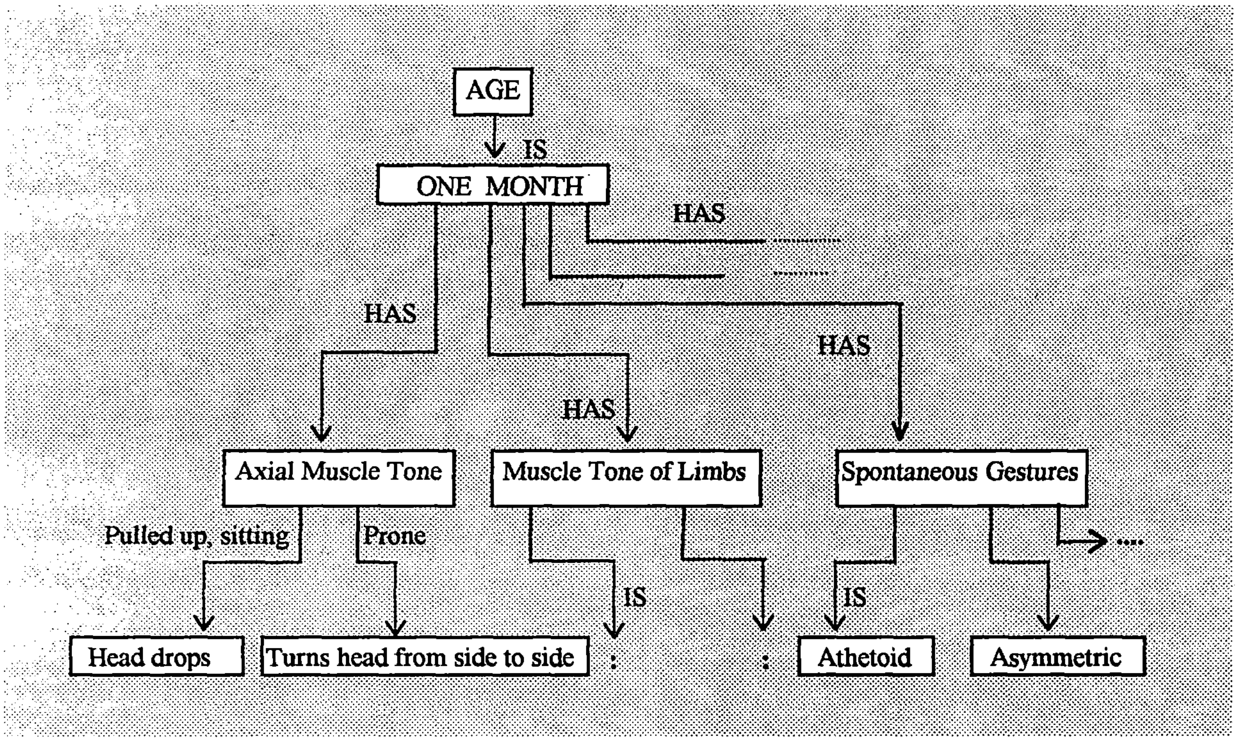

Fig. 6. 19 Semantic networks

- **Rules**

In the form of rules the knowledge is represented as :

Rule 1.

If the baby of age between 1 day and 1 month drops his / her head when he / she is pulled up sitting and turns head from side to side when he / she is proned **Then** the axial muscle tone is normal.

:

Rule 8.

If the baby of age between 1 day and 1 month sleeps 21 hours and takes 5 meals **Then** rhythms are normal.

- **Frames**

The knowledge of **Appendix A** may be represented using frames as :

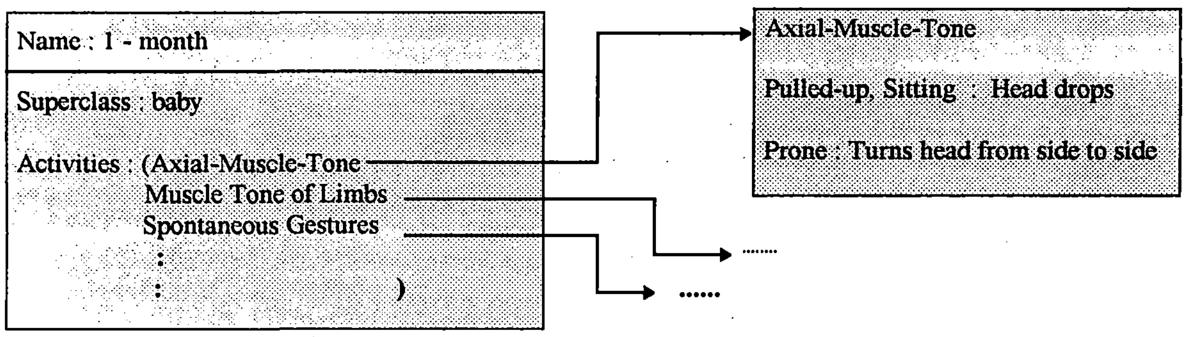

Fig. 6.20 Frame description of baby

- **Object-Oriented approach**

The knowledge of **Appendix A** may be represented using O-O approach as :

```
Class name : age
Super class : baby
Instance variables
  Head :
  Limbs :
  Gestures :
  Reflexes :
  Vision :
  Pulled-up-sitting :
  Prone :
  :
Instance methods
  Axial-muscle-tone( ) :
        begin
           message(Pulled-up-sitting,head-drops);
           message(Prone,head-turns-side-to-side);
           :
           end.
  :
  :
```

Fig. 6.21 object - oriented representation

6.6. **Discussions**

In this chapter, we have tried to analyse the relative suitability of different KR schemes from the viewpoint of an expert system designer for the problem domain of child growth and development (CGD). Our analysis finds O-O approach more suitable for the problem domain of CGD for North-Bengal districts of India. Hopefully, this analysis will be useful to other expert system designers.

While above analysis and consequent results might lead one to believe that the object-oriented paradigm is a panacea for all the woes of knowledge engineering / abstraction / representation, the paradigm does have some drawbacks [29] :

- One of object-oriented technology's disadvantage is its long learning curve. The classical developers have to devote several months before they are skilled enough to start a project.

- Second problem may be that is expected in the initial stages of any relatively new technology is the unavailability of robust and reliable tools such as AI-language or a shell. However, at present, there are some ES-shells using object technology e.g. Level5 (Object) are coming into the market.

- The third problem may come from the very nature of abstraction. The reliability of the abstraction layer(s) should be sufficiently high so that there should not be any bug with these layer(s). These bugs are rarely trapped by the application layer due to the shielding property of abstraction. A careful design is, obviously, required to overcome this problem.

At this stage, there is no doubt that O-O technology should certainly assist us : (i) in developing a complex system; (ii) in maintaining the system; and (iii) in modifying the knowledge base of a system.

References

1. N. J. Nilsson. Principles of Artificial Intelligence. Narosa Publishing House; New Delhi, 1990.

2. Elaine Rich and Kevin Knight. Artificial Intelligence (2nd ed.). Tata McGraw-Hill; New Delhi,1991.

3. Henry C. Mishkoff. Understanding Artificial Intelligence. H. W. Sams and Co.; 1985, 9-10.

4. Efraim Turban. Expert Systems and Applied Artificial Intelligence. Macmillan Publishing Company; New York, 1992.

5. Eugene Charniak and Drew McDermott. Introduction to Artificial Intelligence. Addison-Wesley; 1985.

6. K. S. Leung. and M. H. Wong. An expert system shell using structured knowledge : An object-oriented approach. IEEE Computer; vol. 23, no.3, March 1990, 38-47.

7. D. Dutta Majumder. Artificial Intelligence and Expert Systems : its role in Industrial System Development. National Symp. on Elec. and Appl; September 1989.

8. George F. Luger and William A. Stubblefield. Artificial Intelligence and the design of expert systems. The Benjamin / Cummings Publishing Co.; Inc., 1989, 34-35.

9. M. Van et al. Emycin manual. Tech. report. Heuristic programming project; Stanford University, 1981.

10. F. R. Kehler. The role of frame-based representation in reasoning. Comm. ACM; vol.28, no.9, Sept. 1985, 904-920.

11. D. A. Waterman. A Guide to expert systems. Addision-Wesley, Reading, Mass; 1986.

12. Marvin Minsky. A framework for representing knowledge. The Psychology of computer vision, (ed.), P. H. Winston. McGraw Hill; New York, 1975.

13. Roger C. Schank and Peter G. Childers. The Cognitive Computer. Addison-Wesley; 1984.

14. A. Goldberg and D. Robson. Smalltalk-80 : The language and its Implementation. Addison-Wesley; NY, 1983.

15. R. S. Pressman. Software Engineering : A practitioner's approach. 3rd edn., McGraw-Hill; Inc., 1992.

16. Object-Oriented Requirements Analysis (course notebook). EVB Software Engineering, 1989.

17. K. Nygaard. Basic concepts in object-oriented programming. Sigplan Notices; vol.21, no. 10, October 1986, 117.

18. O. Dahl and K. Nygaard. Simula - A language for programming and description of discrete event systems. Introduction and Users Manual. Norwegian Computing Centre, Oslo, Norway, 1967.

19. J. Durkin. Expert Systems : Catalog of Applications. Intelligent Computer Systems; Inc., Akron, Ohio, 1993.

20. J. Durkin. Expert Systems : A view of the field. IEEE Expert; April 1996, 56-63.

21. W. B. Gevarter, The nature and evaluation of commercial expert system building tools. IEEE Computer; vol.20 no.5, 1987, 24-41.

22. F. Hayes-Roth. Invited Talk. IEEE Compcon; San Francisco, CA, February 1987.

23. A. Synder. Encapsulation and inheritance in object-oriented programming languages. Proc. OOPSLA, ACM; New York, 1986, 38-45.

24. James L. Alty. Expert System Building Tools. Topics in Expert System Design; G. Guida and C. Tasso,(ed.), Elsevier Science Publishers B. V.; (North-Holland), 1989, 193.

25. F. Hayes-Roth, Donald A. Waterman and Douglas B. Lenat. An overview of Expert Systems. Building an Expert System, Part II; 26.

26. P. H. Winston and B. K. Horn. LISP. (2nd ed.). Addison-Wesley, Reading, Mass.; 1985.

27. A. K. Saha, P. Mondal and R. K. Samanta. A prototyped-object-oriented expert system for child growth and development. Modelling, Measurement and Control; C, AMSE Press, France, vol.31, no.2, 1992, 13-24.

28. P. Mondal, A. K. Saha and R. K. Samanta. On an expert system with child growth and development. Advances in Modelling and Analysis; B, AMSE Press, France. vol.26, no.1,1993,13-28.

29. R. Gupta, W. H. Cheng, R. Gupta, I. Hardonag and M. A. Breuer. An Object-Oriented VLSI CAD Framework : A case study in rapid prototyping. IEEE Computer; May 1989, 28-37.