Chapter-7

7. EXPERT SYSTEMS

1. Overview of an Expert Systems:

An Expert System (ES) is a set of programs that manipulates some encoded knowledge to solve problems in a specialized domain that normally requires human expertise. (OR)

An ES is a computer application that solves complicated problems that would otherwise require human expertise.

→An expert system's knowledge is obtained from expert sources and coded in a form suitable for the system to use in its inference or reasoning processes. The expert knowledge must be obtained from specialists or other sources of expertise, such as texts, journal, articles and databases. This type of knowledge usually requires much training and experience in some specialized field such as medicine, geology, system configuration, or engineering design. Once a sufficient body of expert knowledge has been acquired, it must be encoded in some form, loaded into a knowledge base, then tested, and refined continually throughout the life of the system

Characteristic Features of Expert Systems:

Expert systems differ from conventional computer system in several important ways

1. Expert systems use knowledge rather than data to control the solution process. Much of the knowledge used in heuristic in nature rather than algorithmic.

2. The knowledge is encoded and maintained as an entity separate from the control program. As such, it is not complicated together with the control program itself. This permits the incremental addition and modification of the knowledge base without recompilation of the control programs. Furthermore, it is possible in some cases to use different knowledge bases with the same control programs to produce different types of expert systems. Such systems are known as expert system shells since they may be loaded with different knowledge bases.

3. Expert systems are capable of explaining how a particular conclusion was reached, and why requested information is needed during a consultation. This is

important as it gives the user a chance to assess and understand the systems reasoning ability, thereby improving the user's confidence in the system.

4. Expert systems use symbolic representations for knowledge and perform their inference through symbolic computations that closely resemble manipulations of natural language.

5. Expert systems often reason with **Metaknowledge**, that is, they reason with knowledge about themselves, and their own knowledge limits and capabilities.

Applications of an Expert Systems:

- Different types of medical diagnoses (internal medicine, pulmonary diseases, inflection blood diseases, and so on).
- Diagnosis of complex electronic and electro-mechanical system.
- Diagnosis of diesel electric locomotion system.
- Diagnosis of software development projects.
- Planning experiment in biology, chemistry, and molecular genetics, forecasting crop damage
- Identification of chemical compound structures and chemical compounds.
- Location of faults in computer and communication systems.
- Scheduling of customer orders, jobs shop productions, computer resources for operating systems, and various manufacturing tasks.
- Evaluation of loan applicants for lending institutions.
- Assessment of geological structures from dip meter logs.
- Analysis of structural systems for design or as a result of earthquake damage.
- The optimal configuration of components to meet given specifications for complex system (like computers or manufacturing facilities).
- Estate planning for minimal taxation and other specific goals.
- Stock and bond portfolio selection and management.
- The design of very large scale integration (VLSI) systems.
- Numerous military applications ranging from battlefield assessment to ocean surveillance.
- Numerous applications related to space planning and exploration.
- Numerous areas of law including civil case evaluation, product liability, assault and battery, and general assistance in locating different law precedents.

- Planning curricula for students.
- Teaching students specialized tasks (like trouble shooting equipment faults).

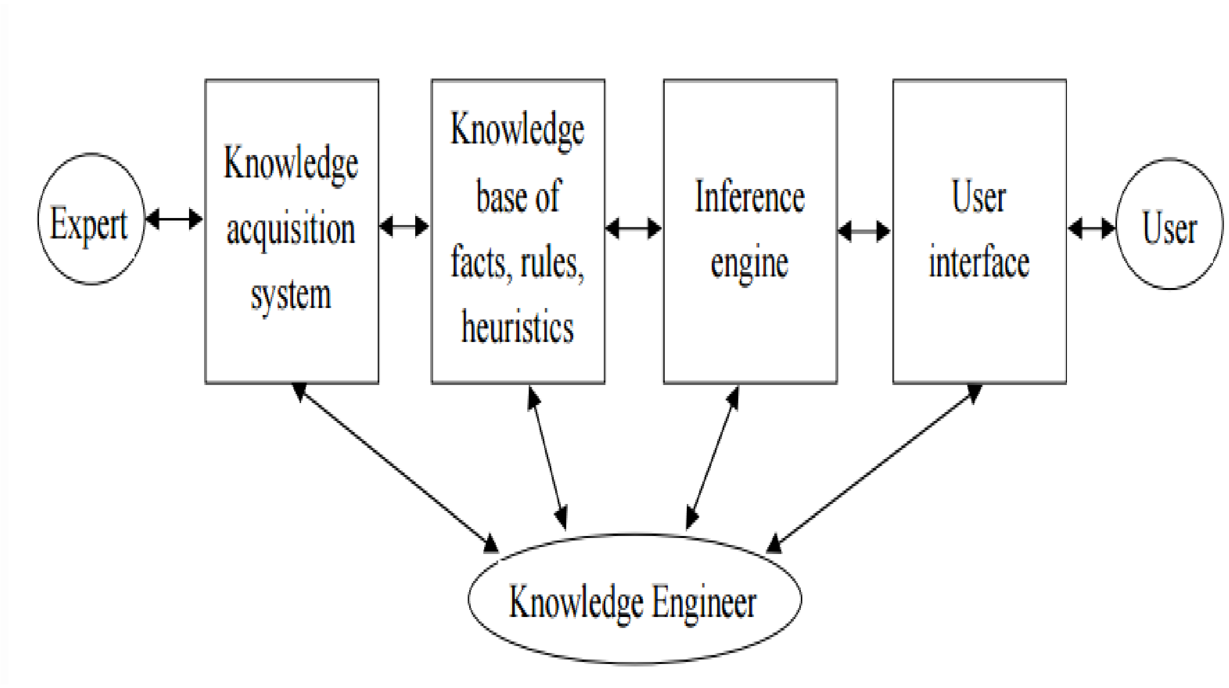
Components of a typical ES are:

- User Interface or I/O Interface: The user or I/O interface facility must accept information from the user and translate it into a form acceptable to the remainder of the system,or accept information from the system and convert it in to a form that can be understood by the user.
- Explanation Module: This provides the user with an explanation of the reasoning process when requested.The explanation consists of an identification of the steps in the reasoning process and a justification for each step.
- Knowledge Base (KB): It represents a storehouse of knowledge primitives.i.e. the knowledge base contains facts and rules about some specialized knowledge domain.
- Inference Engine (IE): It accepts user I/P's queues and responds to questions through the I/O interface and uses this dynamic information together with the static knowledge (rules and facts) stored in the KB.
- Learning Module or Knowledge Update: This is not a common component of ES. When they are provided, they are used to assist in building and refining the KB. The system in effect learns from experience and so is self updating or self learning.

.

2. Architecture of an Expert Systems:

The process of building expert systems is often called knowledge engineering. The knowledge engineer is involved with all components of an expert system:


Expert
Knowledge acquisition system
Knowledge base of facts, rules, heuristics
Inference engine
User interface
User
Knowledge Engineer

Building expert systems is generally an iterative process. The components and their interaction will be refined over the course of numerous meetings of the knowledge engineer with the experts and users. We shall look in turn at the various components.

<u>1 Knowledge Acquisition:</u>
The knowledge acquisition component allows the expert to enter their knowledge or expertise into the expert system, and to refine it later as and when required.
Historically, the knowledge engineer played a major role in this process, but automated systems that allow the expert to interact directly with the system are becoming increasingly common.

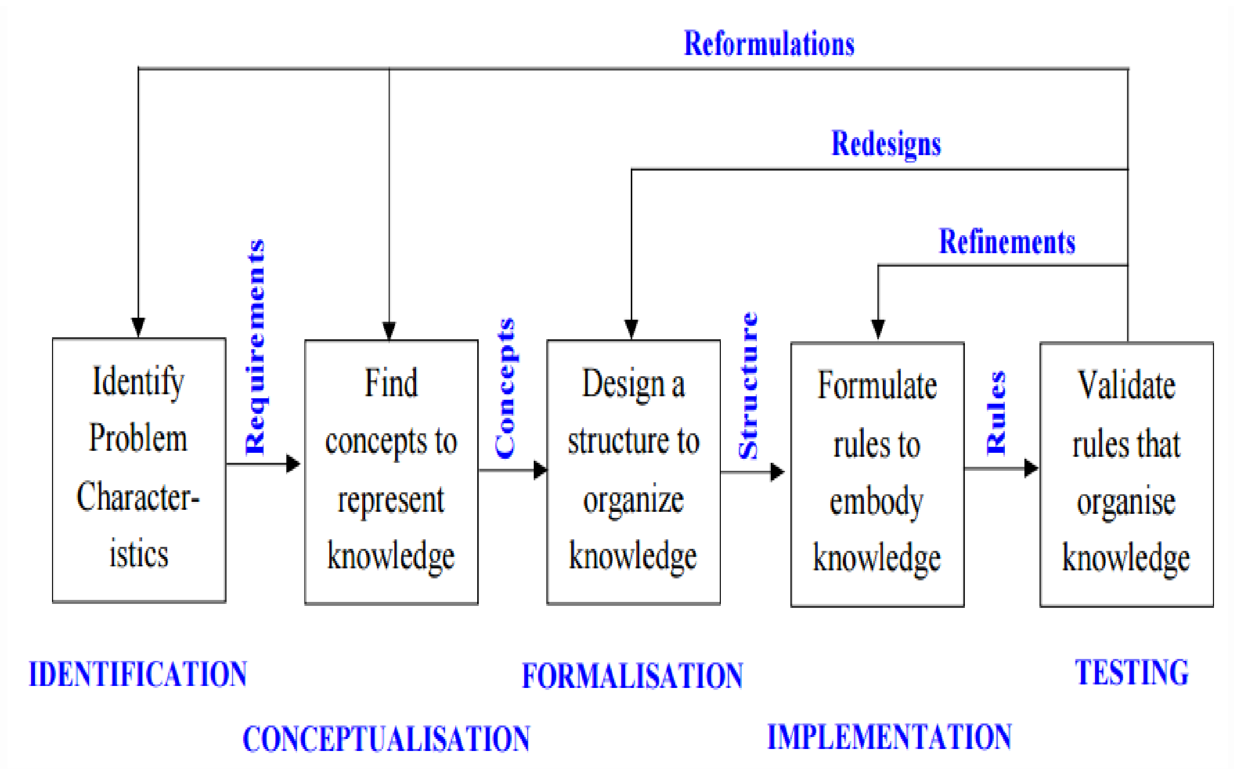
→The knowledge acquisition process is usually comprised of three principal stages:
1. Knowledge elicitation is the interaction between the expert and the knowledge Engineer/program to elicit the expert knowledge in some systematic way.
2. The knowledge thus obtained is usually stored in some form of human friendly intermediate representation.
3. The intermediate representation of the knowledge is then compiled into an executable form (e.g. production rules) that the inference engine can process.
In practice, much iteration through these three stages is usually required!

<u>2 Knowledge Elicitation:</u>

→The knowledge elicitation process itself usually consists of several stages:
1. Find as much as possible about the problem and domain from books, manuals, etc. In particular, become familiar with any specialist terminology and jargon.
2. Try to characterize the types of reasoning and problem solving tasks that the system will be required to perform.
3. Find an expert (or set of experts) that is willing to collaborate on the project. Sometimes experts are frightened of being replaced by a computer system!
4. Interview the expert (usually many times during the course of building the system). Find out how they solve the problems your system will be expected to solve. Have them check and refine your intermediate knowledge representation.
This is a time intensive process, and automated knowledge elicitation and machine learning techniques are increasingly common modern alternatives.

3 Stages of Knowledge Acquisition:

The iterative nature of the knowledge acquisition process can be represented in the following diagram.


Reformulations
Redesigns
Refinements
Identify Problem Character-istics
Requirements
Find concepts to represent knowledge
Concepts
Design a structure to organize knowledge
Structure
Formulate rules to embody knowledge
Rules
Validate rules that organise knowledge
IDENTIFICATION
FORMALISATION
TESTING
CONCEPTUALISATION
IMPLEMENTATION

4 Levels of Knowledge Analysis:

Knowledge identification: Use in depth interviews in which the knowledge engineer encourages the expert to talk about how they do what they do. The knowledge engineer should understand the domain well enough to know which objects and facts need talking about.

Knowledge conceptualization: Find the primitive concepts and conceptual relations of the problem domain.
Epistemological analysis: Uncover the structural properties of the conceptual knowledge, such as taxonomic relations (classifications).
Logical analysis: Decide how to perform reasoning in the problem domain. This kind of knowledge can be particularly hard to acquire.
Implementation analysis: Work out systematic procedures for implementing and testing the system.
Capturing Tacit/Implicit Knowledge
One problem that knowledge engineers often encounter is that the human experts use tacit/implicit knowledge (e.g. procedural knowledge) that is difficult to capture.

→There are several useful techniques for acquiring this knowledge:
1. **Protocol analysis**: Tape-record the expert thinking aloud while performing their role and later analyze this. Break down their protocol/account into the smallest atomic units of thought, and let these become operators.
2. **Participant observation**: The knowledge engineer acquires tacit knowledge through practical domain experience with the expert.
3. **Machine induction**: This is useful when the experts are able to supply examples of the results of their decision making, even if they are unable to articulate the underlying knowledge or reasoning process. Which is/are best to use will generally depend on the problem domain and the expert.

5 Representing the Knowledge:

→We have already looked at various types of knowledge representation. In general, the knowledge acquired from our expert will be formulated in two ways:
1. **Intermediate representation** – a structured knowledge representation that the knowledge engineer and expert can both work with efficiently.
2. **Production system** – a formulation that the expert system's inference engine can process efficiently.
It is important to distinguish between:
1. **Domain knowledge** – the expert's knowledge which might be expressed in the form of rules, general/default values, and so on.
2. **Case knowledge** – specific facts/knowledge about particular cases, including any derived knowledge about the particular cases. The system will have the domain knowledge built in, and will have to integrate this with the different case knowledge that will become available each time the system is used.

3. Different Types of an Expert Systems:

Different Types of Expert Systems are

- **Rule Based Architectures**
- **Frame Based Architectures**
- **Decision Tree based Architectures**
- **Case Based Architectures**
- **Neural Network based Architectures**
- **Black Board Architectures**

PRODUCTION SYSTEM ARCHITECTURES

4. RULE BASED SYSTEM ARCHITECTURES:

This is the most common form of architecture used in expert and other types of knowledge based systems, also called rule based systems. This type of system uses knowledge encoded in the form of production rules. i.e., if..........then rules.

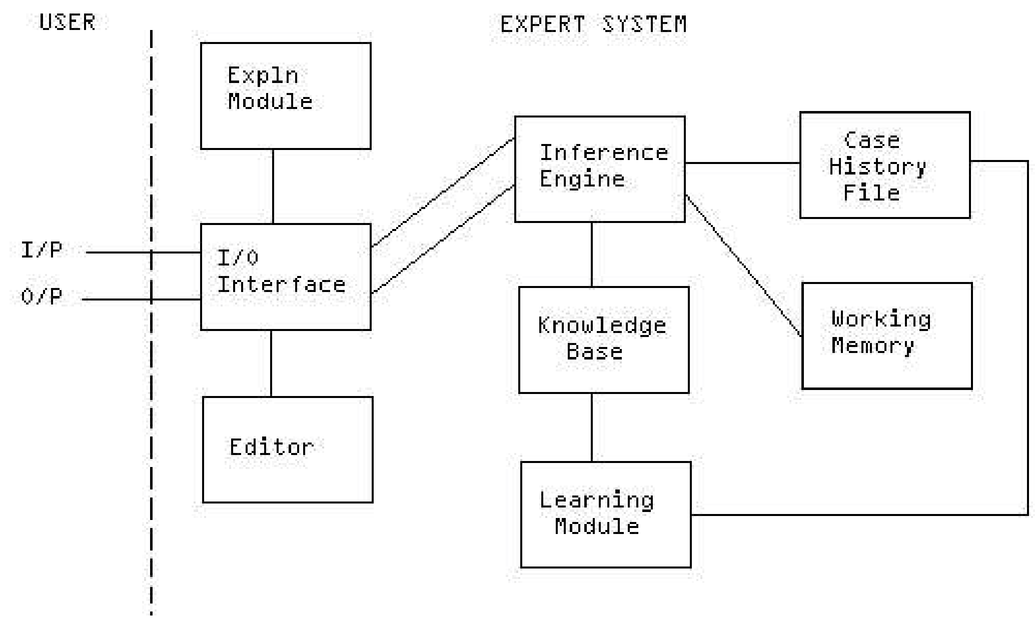
IF : condition1 and condition2 and condition3
THEN : Take action4
IF: This temperature is greater than 200 degrees, and
 The water level is low.
THEN: Open the safety valve.

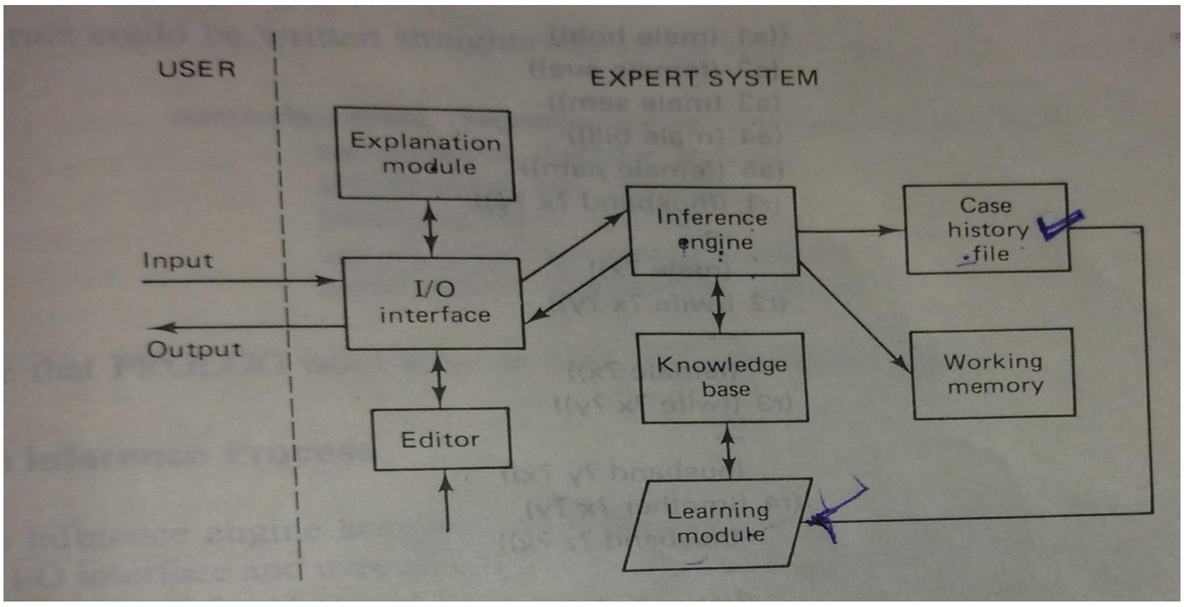
A & B & C & D → E & F

Each rule represents a small chunk of knowledge relating to the given domain of expertise. A number of related rules collectively may correspond to a chain of inferences which lead from some initially known facts to some useful conclusions.

Inference in production systems is accomplished by a process of chaining through the rules recursively, either in a forward or backward direction until a conclusion is reached or until failure occurs.

Block Diagram of a Typical ES





→The main components of a typical expert system are depicted in the above figure,

<u>The Knowledge Base:</u>

The knowledge base contains facts rules about some specialized knowledge domain. An example of a simple knowledge base giving family relationships is illustrated in figure.

```
((a1 (male bob))
 (a2 (female sue))
 (a3 (male sam))
 (a4 (male bill))
 (a5 (female pam))
 (r1 ((husband ?x ?y))
        →
 (male ?x)
 (r2 ((wife ?x ?y))
        →
 (female ?x))
```

```
(r3 ((wife ?x ?y))
     →
     (husband ?y ?x))
(r4 ((mother ?x ?y)
     (husband ?z ?x))
     →
     (father ?z ?y))
(r5 ((father ?x ?y))
     (wife ?z ?x))
     →
     (mother ?z ?y))
(r6 ((husband ?x ?y))
     →
     (wife ?y ?x))
(r7 (( father ?x ?z)
     (mother ?y ?z))
     →
     (husband ?x ?y))
(r8 (( father ?x ?z)
     (mother ?y ?z))
     →
     (wife ?y ?z))
(r9 (( father ?x ?y)
     (father ?y ?z))
     →
     (grand father ?x ?z)))
```

Fig. 7.3 Facts and Rules in Knowledge Base

In PROLOG, rules are written naturally as clauses with both a head and body.

→For example, s rule about a patient's symptoms and the corresponding diagnosis of hepatitis might read in English as the rule

```
  IF : The patient has a chronic disorder, and
       the sex of the patient is female, and
       The age of the patient is less than 30, and
       the patient shows condition A, and
       test B reveals biochemistry condition
THEN:  Conclude the patient's diagnosis is autoimmune-chronic-hepatitis.
```

→This rule could be written straightaway in PROLOG as

Conclude (patient, diagnosis, autoimmune, chronic hepatitis):-

```
    same (patient, disorder, chronic);
    same (patient, sex, female),
    less than (patient, age, 30),
    same (patient, symptom_a, value_a),
    same (patient, biochemistry, value_c).
```

The prolog rules have atmost one conclusion clause.

The inference Process:

→The inference process is carried out recursively in 3 stages:

1. **Match**
2. **Select**
3. **Execute**

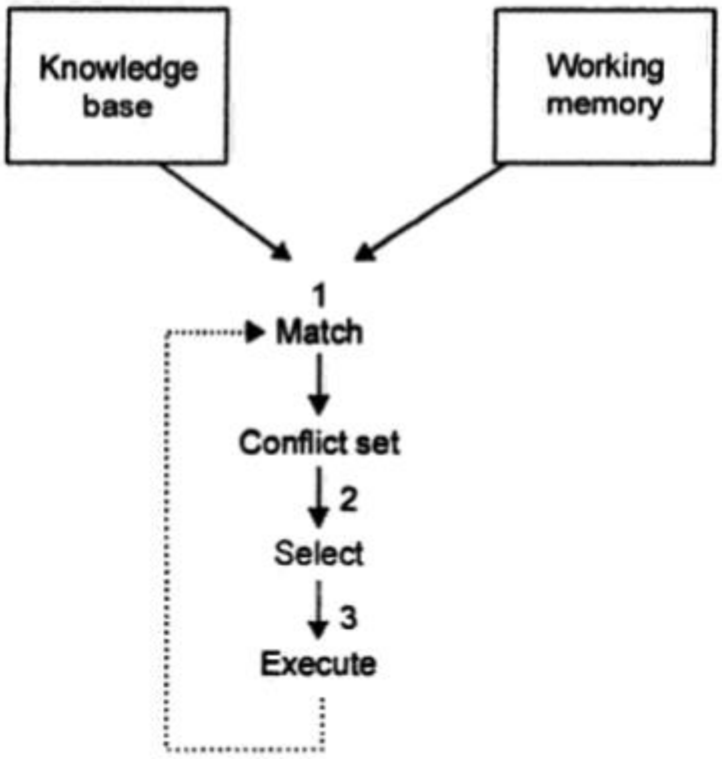
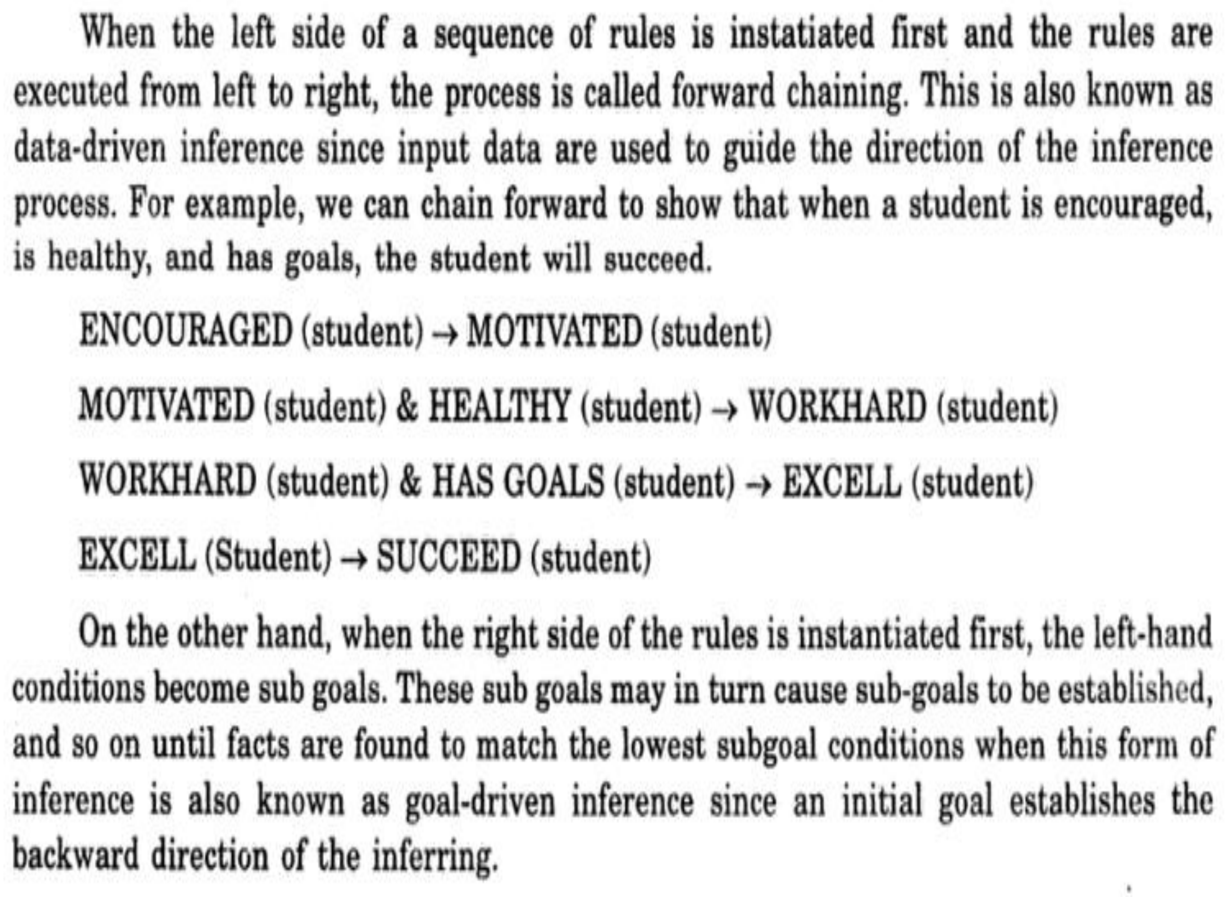


Fig. 7.2 The Production System Inference Cycle

During the match stage, the contents of working memory are compared to facts and rules contained in the knowledge base. When consistent matches are found, the corresponding rules are placed in a conflict set. To find an appropriate and consistent match substitution (instantiations) may be required. Once all the matched rules have been added to the conflict set during a given cycle, one of the rules is selected for execution. The criteria for selection may be most recent use, rule condition specificity, (the number of conjucts on the left), or simply the smallest rule number. The selected rules is then executed and the right-hand side or action port of the rule is then carried out.



When the left side of a sequence of rules is instatiated first and the rules are executed from left to right, the process is called forward chaining. This is also known as data-driven inference since input data are used to guide the direction of the inference process. For example, we can chain forward to show that when a student is encouraged, is healthy, and has goals, the student will succeed.

ENCOURAGED (student) → MOTIVATED (student)

MOTIVATED (student) & HEALTHY (student) → WORKHARD (student)

WORKHARD (student) & HAS GOALS (student) → EXCELL (student)

EXCELL (Student) → SUCCEED (student)

On the other hand, when the right side of the rules is instantiated first, the left-hand conditions become sub goals. These sub goals may in turn cause sub-goals to be established, and so on until facts are found to match the lowest subgoal conditions when this form of inference is also known as goal-driven inference since an initial goal establishes the backward direction of the inferring.

Inference Engine:

➔The inference engine accepts user input queries and responses to questions through the I/O interface. It uses the dynamic information together with the static knowledge stored in the knowledge base. The knowledge in the knowledge base is used to derive conclusions about the current case as presented by the user's input. Inference engine is the module which finds an answer from the knowledge base. It applies the knowledge to find the solution of the problem. In general, inference engine makes inferences by deciding which rules are satisfied by facts, decides the priorities of the satisfied rules and executes the rule with the highest priority. Generally inferring process is carried out recursively in 3 stages like match, select and execute. During the match stage, the contents of working memory are compared to facts and rules contained in the knowledge base. When proper and consistent matches are found, the corresponding rules are placed in a conflict set.

Explaining How or Why:

Explanation Module:

➔The explanation module explains the reasoning of the system to a user.
It provides the user with an explanation of the reasoning process when requested. The credibility of expert system will be established only when it is able to explain "how and why" a particular conclusion is drawn. This explanation increases the belief of user in the expert system.

a) **Explanation(How):** To respond to a how query, the explanation module traces the chain of rules fired during a consolation with the user. This explanation mode can be activated once the process is over. It explains how a fact was deduced by the system and similarly how a rule was/wasn't used. The simplest way to specify this is to explain the rule which allows the deduction. For e.g.

If the system (S) will give information about the parent-child relationship to the user (U) then the followings can be possible.

S: My diagnosis is "A is the father of B"

U: How?

S: The result was obtained by the addition of following facts and rules.

Fact no 11: A is the parent of Hari.

Fact no 15: A is a male.

Fact no 110: X is father of Y:

X is parent of Y, X is male.

So A is the father of B.

b) Explanation (Why)? To respond to a why query, the explanation module must be able to explain why certain information is needed by the inference engine to complete a step in the reasoning process. This mode of explanation is dynamic and is used when the system functions in backward chaining. If the user doesn't understand the reason behind the question. The system is capable of explaining why the question was asked. For example S: Is the following true?

A is the father of B.

U: Why?

S:I need the fact:

A is the father of B to establish the following fact “B is the son of A”.

By using the rule no. 4:

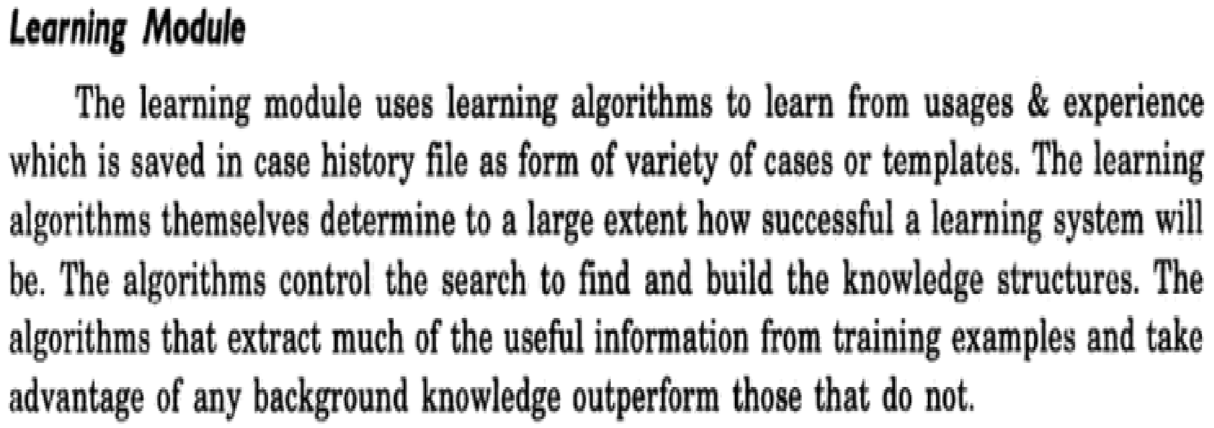
A is the father of B:

B is the son of A.

Building a Knowledge Base:

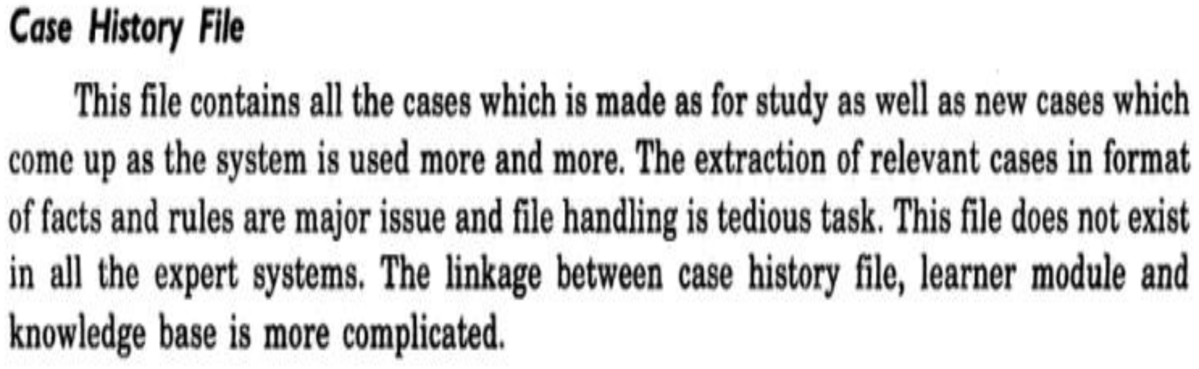
→**Editor** is used for editing knowledge in knowledge data base .If any data is not needed in knowledge database that can be removed from the knowledge data base and if any data is needed for the domain that can be inserted in to the database.

→ **Knowledge Engineering:** The primary people involved in building an expert system are the knowledge engineer, the domain expert and the end user. Once the knowledge engineer has obtained a general overview of the problem domain and gone through several problem solving sessions with the domain expert, he/she is ready to begin actually designing the system, selecting a way to represent the knowledge, determining the search strategy (backward or forward) and designing the user interface. After making complete designs, the knowledge engineer builds a prototype. The prototype should be able to solve problems in a small area of the domain. Once the prototype has been implemented, the knowledge engineer and domain expert test and refine its knowledge by giving it problems to solve and correcting its disadvantages.



Learning Module

The learning module uses learning algorithms to learn from usages & experience which is saved in case history file as form of variety of cases or templates. The learning algorithms themselves determine to a large extent how successful a learning system will be. The algorithms control the search to find and build the knowledge structures. The algorithms that extract much of the useful information from training examples and take advantage of any background knowledge outperform those that do not.



Case History File

This file contains all the cases which is made as for study as well as new cases which come up as the system is used more and more. The extraction of relevant cases in format of facts and rules are major issue and file handling is tedious task. This file does not exist in all the expert systems. The linkage between case history file, learner module and knowledge base is more complicated.

PRODUCTION SYSTEM ARCHITECTURES

5. FRAME ARCHITECTURES:

→Frames are structured sets of closely related knowledge such as on object or concept name, the objects main attribute and there corresponding values and possibly some attached procedures (if needed, if added, if-removed procedures). The attributes, values and procedures are stored in specified slots and facets of the frame. Individual frames are linked together in a network much like nodes in an associative network, namely, properly inheritance and default reasoning several expert systems have been constructed with frame architectures, and a number of building tools which create and manipulate frame structured systems have been developed.

Thus frames have many of the features of associative networks like property inheritance and default reasoning.

→An example of frame based system is the PIP systems (Present Illness Program) developed at M.I.T during the late 1970's and 1980's

→This system was used to diagnose patients using low costs, easily obtained information, the type of information obtained by a general practitioner during an office examination.

→The medical knowledge in PIP is organized in frame structures, where each frame is obtained of categories of slots with names such as

Typical findings

Logical decision criteria

Complementary relations to other frames

Differential diagnosis

Scoring

→The patient findings are matched against frames, and when a close match is found, a trigger status occurs.

6. Decision Tree Based Architectures:

→Knowledge for Expert System may be stored in the form of a decision tree when the knowledge can be structured in a top to bottom manner. For example, the identification of objects(equipment faults, physical objects, diseases, and the like) can be made through a decision tree structure. Initial & Intermediate nodes in the tree correspond to object attributes, and terminal nodes correspond to the identities of objects. Attribute values for an object determine a path to a leaf node in tree which contains the objects identification. Each object attribute corresponds to a non-terminal node in the tree and each branch of the decision tree corresponds to an attribute value or set of values.

→A segment of a decision tree knowledge structure taken from an expert system used to identify objects such as liquid chemical waste products is illustrated in the following **figure.** Each node in the tree corresponds to the identifying attribute such as molecular weight, boiling point, burn test color, or solubility test results**.**

→The Knowledge base, which is the decision tree for an identification system can be constructed with a special tree building editor or with a learning module.

→New nodes or branches can be added to the tree when additional attributes are needed to further discriminate among new objects. As the system gains experience the values associated with the branches can be modified for more accurate results.

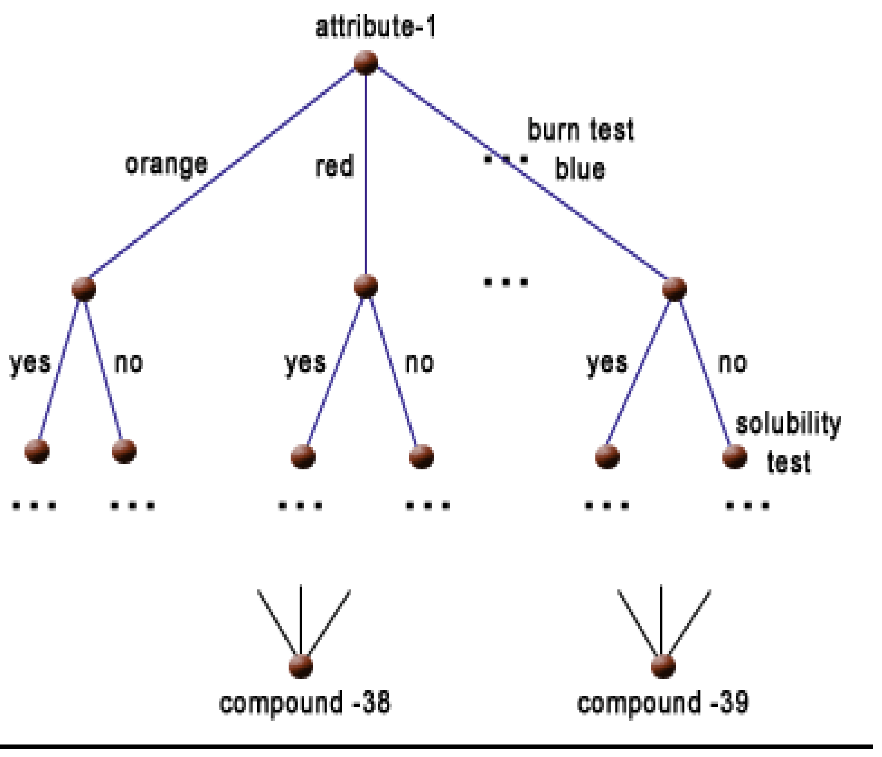


Fig. A segment of a decision tree structure.

7. Blackboard System Architectures:

→Blackboard architectures refer to a special type of knowledge based system which uses a form of opportunistic reasoning. This differs from pure backward or pure forward chaining in production system in that either direction may be choose dynamically at each stage in the problem solution process other reasoning methods may also be used.

→Blackboard systems are composed of 3 functional components as depicted in **figure.**

1. Knowledge Sources

2. Blackboard

3. Control Information

Example: HEARSAY a speech understanding system.

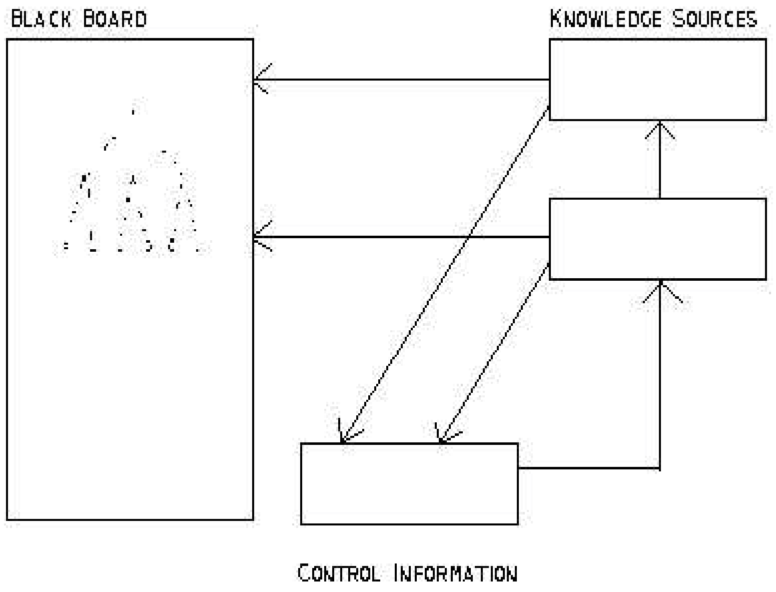


Fig. Components of Blackboard systems.

1. Knowledge Sources:

→There are number of knowledge sources which are separate and independent sets of coded knowledge. Each knowledge source may be thought of as a specialist in some limited area needed to solve a given subset of problems. The source may contain the knowledge in the form of procedures, rules, or other schemes'

2. Blackboard:

→A globally accessible data base structure called blackboard, contains the current problem state and information needed by the knowledge sources (input data, partial solutions, control data, alternatives, and final solutions). The knowledge sources make changes to the backboard data that incrementally leads to solutions. Communication and interaction between the knowledge sources takes place solely through the blackboard.

3. Control Information:

→Control information may be containing within the sources, on the black board, or possibly in a separate module. The control knowledge monitors changes to the blackboard and determines what the intermediate focus of attention should be in solving the problem.

8. Neural Network based Architectures:

Neural networks are large networks of simple processing elements or nodes which process information dynamically in response to external inputs. The nodes are simplified models of neurons. The knowledge in a neural network is distributed throughout the network in the form of internode connections and weighted links which form the inputs to the nodes. The link weights serve to enhance or inhibit the input stimuli values which are then added together at the nodes. If the sum of all the inputs to a node exceeds some threshold value T, the node executes and produces an output which is passed on to other nodes or is used to produce some output response. In the simplest case, no output is produced if the total input is less than T. In more complex models, the output will depend on a nonlinear activation function.

Neural networks were originally inspired as being models of the human nervous system. They are greatly simplified models to be sure (neurons are known to be fairly complex processors). Even so, they have been shown to exhibit many "intelligent" abilities, such as learning, generalization, and abstraction.

A single node is illustrated in Figure 7.6. The inputs to the node are the values x_1, $x_2, \ldots, x_n$, which typically take on values of –1, 0, 1, or real values within the range (–1, 1). The weights $w_1, w_2, \ldots, w_n$, correspond to the synaptic strengths of a neuron. They serve to increase or decrease the effects of the corresponding x_i input values. The sum of the products x_i, w_i, $i = 1, 2, \ldots, n$, serve as the total combined input to the node. If this sum is large enough to exceed the threshold amount T, the node fires, and produces an output y, an activation function-value placed on the node's output links. This output may then be the input; other nodes or the final output response from the network.

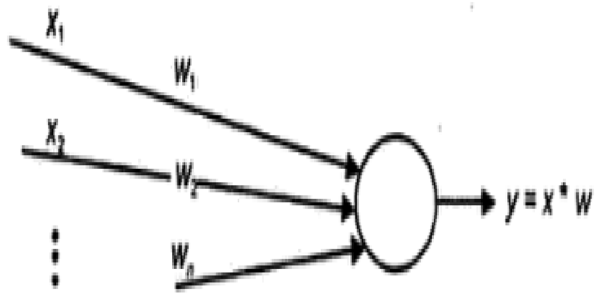
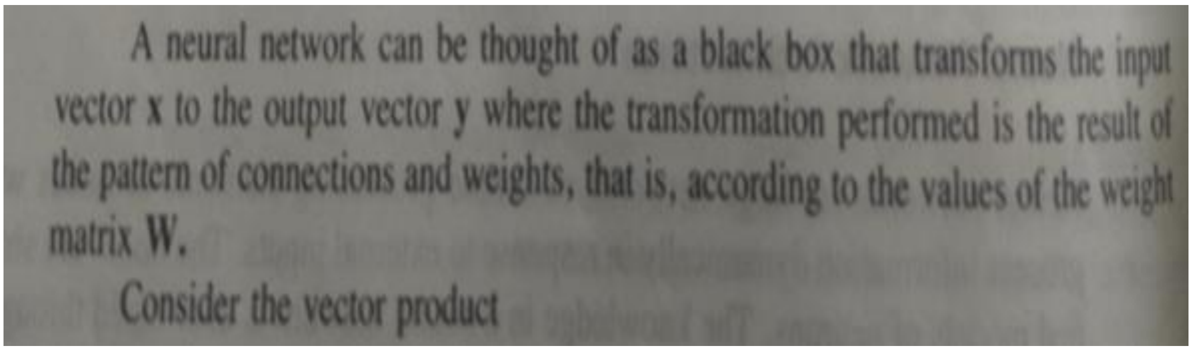


Fig. 7.6 Model of a Single Neuron

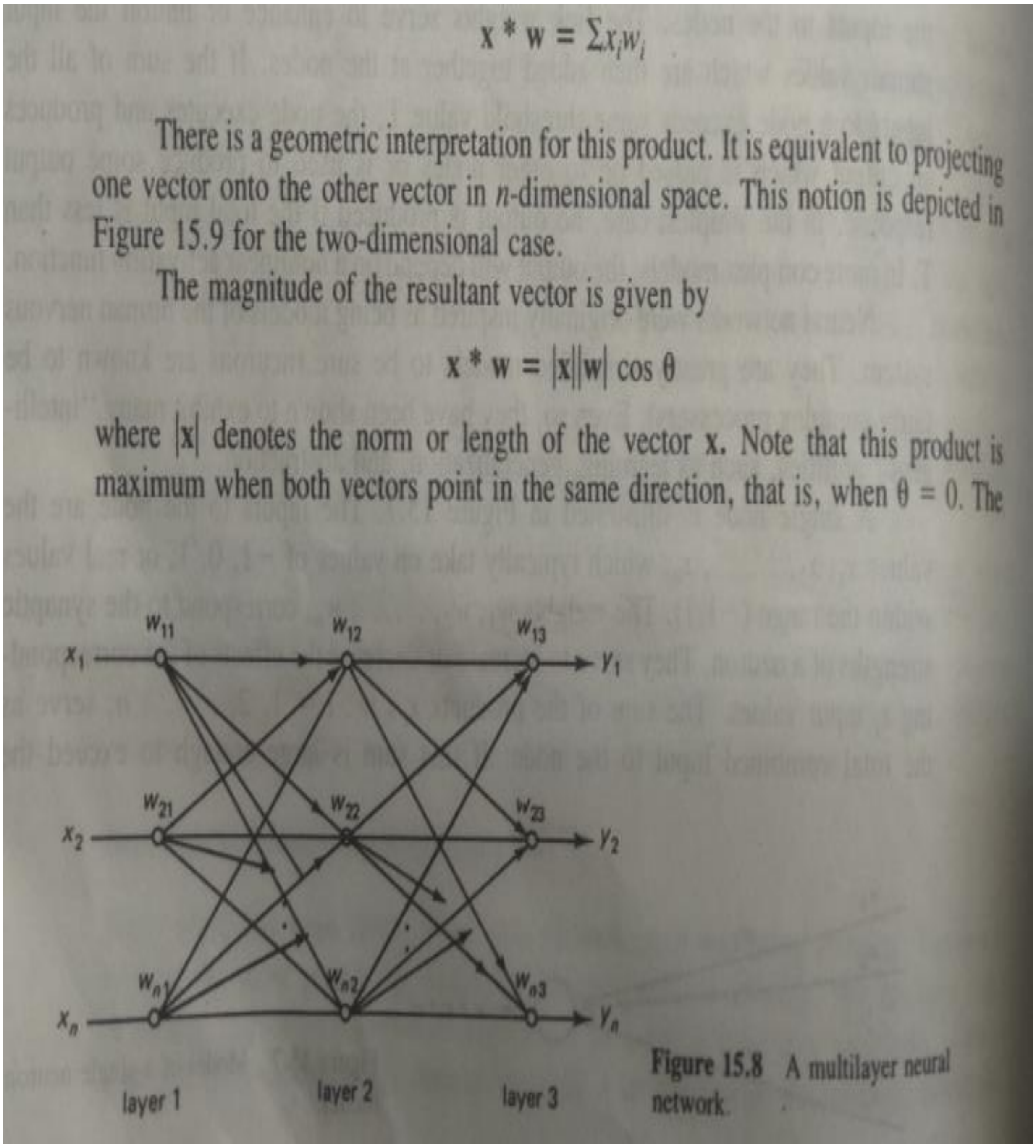
Figure 7.7 illustrates three layers of a number of interconnected nodes. The first layer serves as the input layer, receiving inputs from some set of stimuli. The second layer (called the hidden layer) receives inputs from the first layer and produces a pattern of inputs to the third layer, the output layer. The pattern of outputs from the final layer are the network's responses to the input stimuli patterns. Input links to layer j ($j = 1, 2, 3$) have weights w_{ij} for $i = 1, 2, \ldots, n$.

General multilayer networks having n nodes (number of rows) in each of m layers (number of columns of nodes) will have weights represented as an $n \times m$ matrix **W**. Using this representation, nodes having no interconnecting links will have a weight value of zero. Networks consisting of more than three layers would, of course, be correspondingly more complex than the network depicted in Figure 7.7.



A neural network can be thought of as a black box that transforms the input vector **x** to the output vector **y** where the transformation performed is the result of the pattern of connections and weights, that is, according to the values of the weight matrix **W**.

Consider the vector product



$$\mathbf{x} * \mathbf{w} = \sum x_i w_i$$

There is a geometric interpretation for this product. It is equivalent to projecting one vector onto the other vector in *n*-dimensional space. This notion is depicted in Figure 15.9 for the two-dimensional case.

The magnitude of the resultant vector is given by

$$\mathbf{x} * \mathbf{w} = |\mathbf{x}||\mathbf{w}| \cos \theta$$

where $|\mathbf{x}|$ denotes the norm or length of the vector **x**. Note that this product is maximum when both vectors point in the same direction, that is, when $\theta = 0$. The

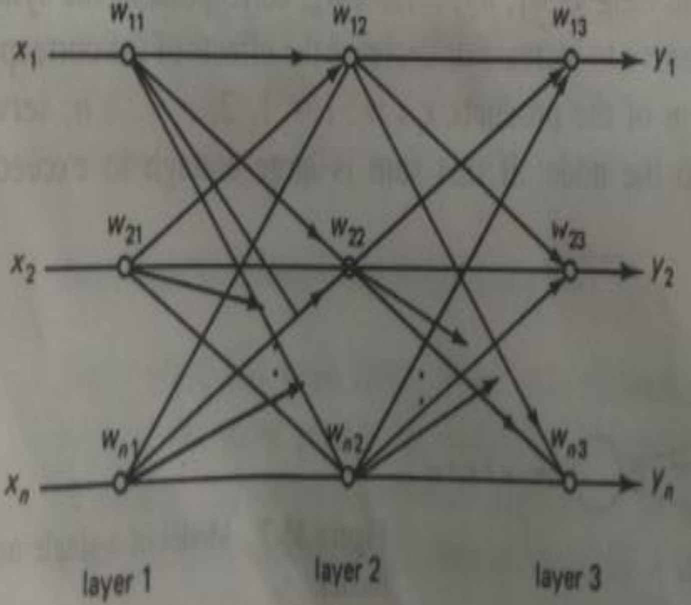


Figure 15.8 A multilayer neural network.

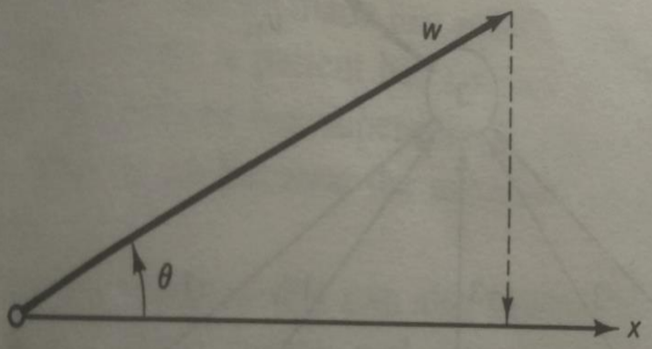


Figure 15.9 Vector multiplication is like vector projection.

product is a minimum when both point in opposite directions or when θ = 180 degrees. This illustrates how the vectors in the weight matrix **W** influence the inputs to the nodes in a neural network.

7.3.8 A Neural Net Expert System

An example of a simple neural network diagnostic expert system has been described by Stephen Gallant (1988). This system diagnoses and recommends treatments for acute sarcophagal disease. The system is illustrated in Figure 7.9. From six symptom variables $u_1, u_2, \ldots, u_6$, one of two possible diseases can be diagnosed, u_7 or u_8. From the resultant diagnosis, one of three treatments, u_9, u_{10}, or u_{11} can then be recommended.

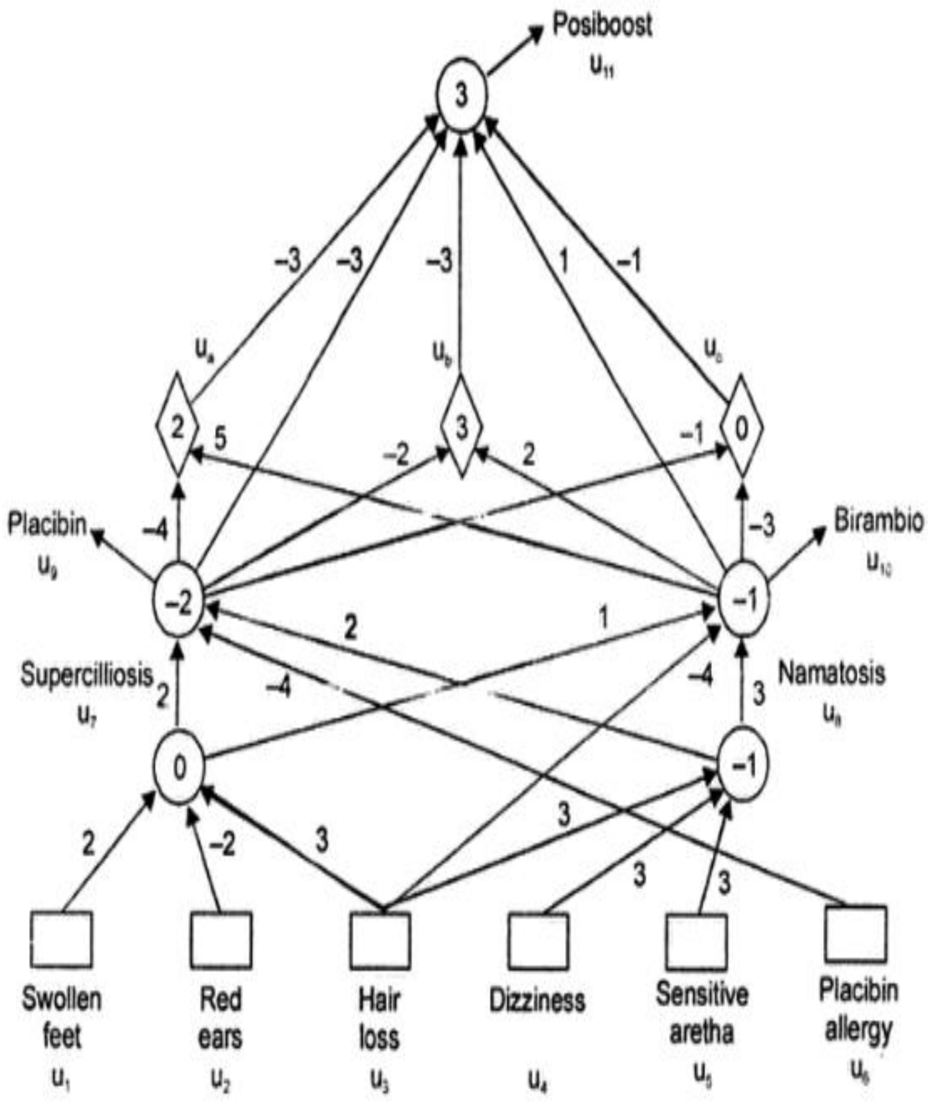


Fig. 7.9 A simple Neural Network Expert System. (From S. 1. Gallant. ACM Communications, Vol. 31, No. 2, p. 152, 1988. By permission.)

4. Knowledge Acquisition and validation Techniques:

Knowledge acquisition is the process used to define the rules and ontologies required for a knowledge-based system. The phrase was first used in conjunction with expert systems to describe the initial tasks associated with developing an expert system, namely finding and interviewing domain experts and capturing their knowledge via rules, objects, and frame-based ontologies.

Expert systems were one of the first successful applications of artificial intelligence technology to real world business problems.[1] Researchers at Stanford and other AI laboratories worked with doctors and other highly skilled experts to develop systems that could automate complex tasks such as medical diagnosis. Until this point computers had mostly been used to automate highly data intensive tasks but not for complex reasoning. Technologies such as inference engines allowed developers for the first time to tackle more complex problems.

As expert systems scaled up from demonstration prototypes to industrial strength applications it was soon realized that the acquisition of domain expert knowledge was one of if not the most critical task in the knowledge engineering process. This knowledge acquisition process became an intense area of research on its own. One of the earlier works on the topic used Batesonian theories of learning to guide the process.

One approach to knowledge acquisition investigated was to use natural language parsing and generation to facilitate knowledge acquisition. Natural language parsing could be performed on manuals and other expert documents and an initial first pass at the rules and objects could be developed automatically. Text generation was also extremely useful in generating explanations for system behavior. This greatly facilitated the development and maintenance of expert systems.

A more recent approach to knowledge acquisition is a re-use based approach. Knowledge can be developed in ontologies that conform to standards such as the Web Ontology Language (OWL). In this way knowledge can be standardized and shared across a broad community of knowledge workers. One example domain where this approach has been successful is bioinformatics.

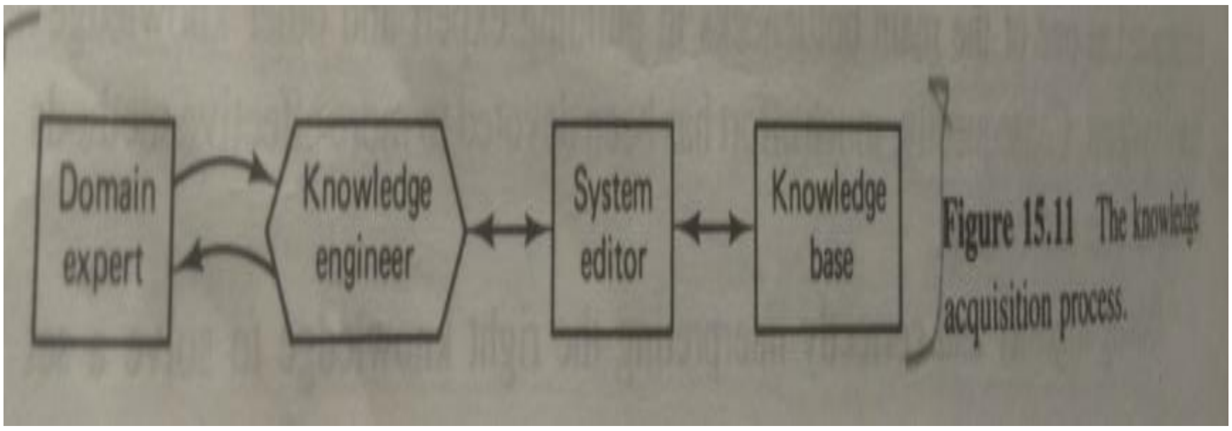


Figure 15.11 The knowledge acquisition process.

2. Knowledge Acquisition

Knowledge acquisition includes the elicitation, collection, analysis, modeling and validation of knowledge.

2.1 Issues in Knowledge Acquisition

The important issues in knowledge acquisition are:

- knowledge is in the head of experts
- Experts have vast amounts of knowledge
- Experts have a lot of tacit knowledge
  - They do not know all that they know and use
  - Tacit knowledge is hard (impossible) to describe
- Experts are very busy and valuable people
- One expert does not know everything
- Knowledge has a "shelf life"

Techniques for Knowledge Acquisition

The techniques for acquiring, analyzing and modeling knowledge are : Protocol-generation techniques, Protocol analysis techniques, Hierarchy-generation techniques, Matrix-based techniques, Sorting techniques, Limited-information and constrained-processing tasks, Diagram-based techniques. Each of these are briefly stated in next few slides.

- **Protocol-generation techniques**
  Include many types of interviews (unstructured, semi-structured and structured), reporting and observational techniques.

- **Protocol analysis techniques**
  Used with transcripts of interviews or text-based information to identify basic knowledge objects within a protocol, such as goals, decisions, relationships and attributes. These act as a bridge between the use of protocol-based techniques and knowledge modeling techniques.

- **Hierarchy-generation techniques**

  Involve creation, reviewing and modification of hierarchical knowledge. Hierarchy-generation techniques, such as laddering, are used to build taxonomies or other hierarchical structures such as goal trees and decision networks. The Ladders are of various forms like concept ladder, attribute ladder, composition ladders.

- **Matrix-based techniques**

  Involve the construction and filling-in a 2-D matrix (grid, table), indicating such things, as may be, for example, between concepts and properties (attributes and values) or between problems and solutions or between tasks and resources, etc. The elements within the matrix can contain: symbols (ticks, crosses, question marks) , colors , numbers , text.

- **Sorting techniques**

  Used for capturing the way people compare and order concepts; it may reveal knowledge about classes, properties and priorities.

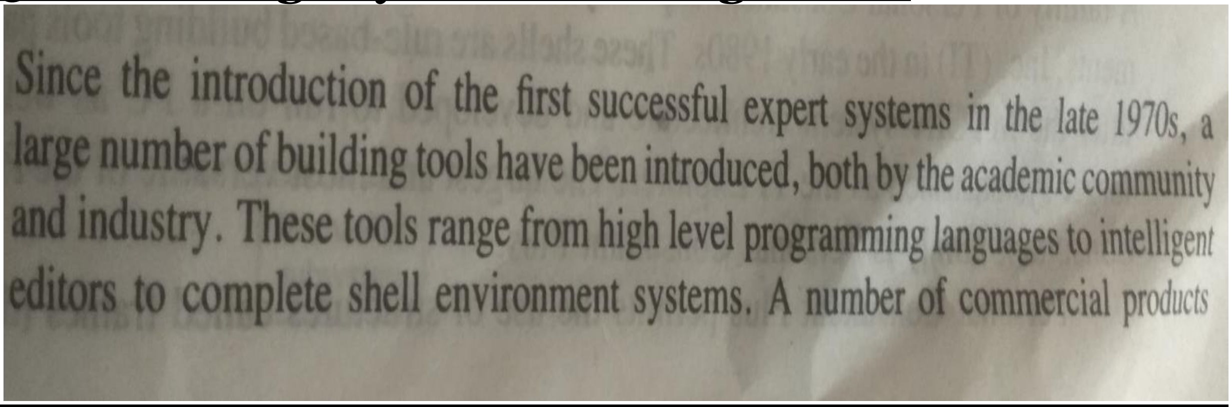
- **Limited-information and constrained-processing tasks**

  Techniques that either limit the time and/or information available to the expert when performing tasks. For example, a twenty-questions technique provides an efficient way of accessing the key information in a domain in a prioritized order.

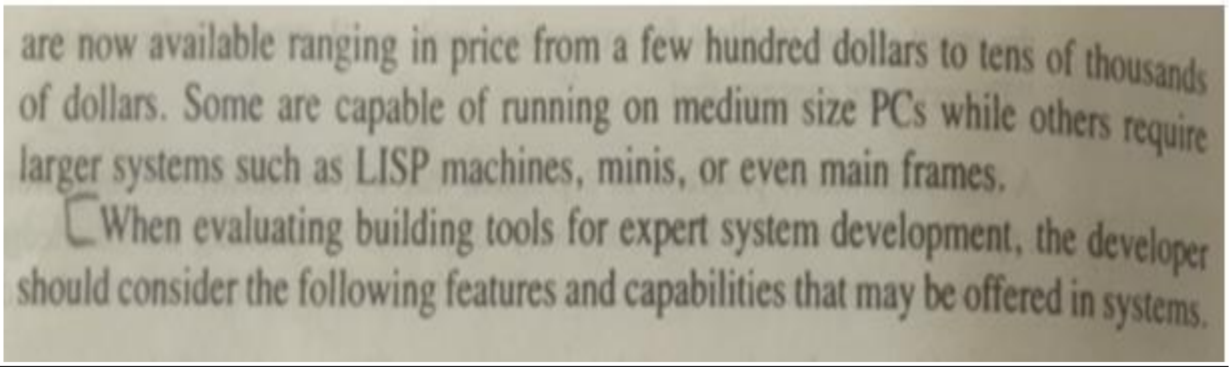
- **Diagram-based techniques**

  Include generation and use of concept maps, state transition networks, event diagrams and process maps. These are particularly important in capturing the "what, how, when, who and why" of tasks and events.

5. Knowledge System Building Tools:

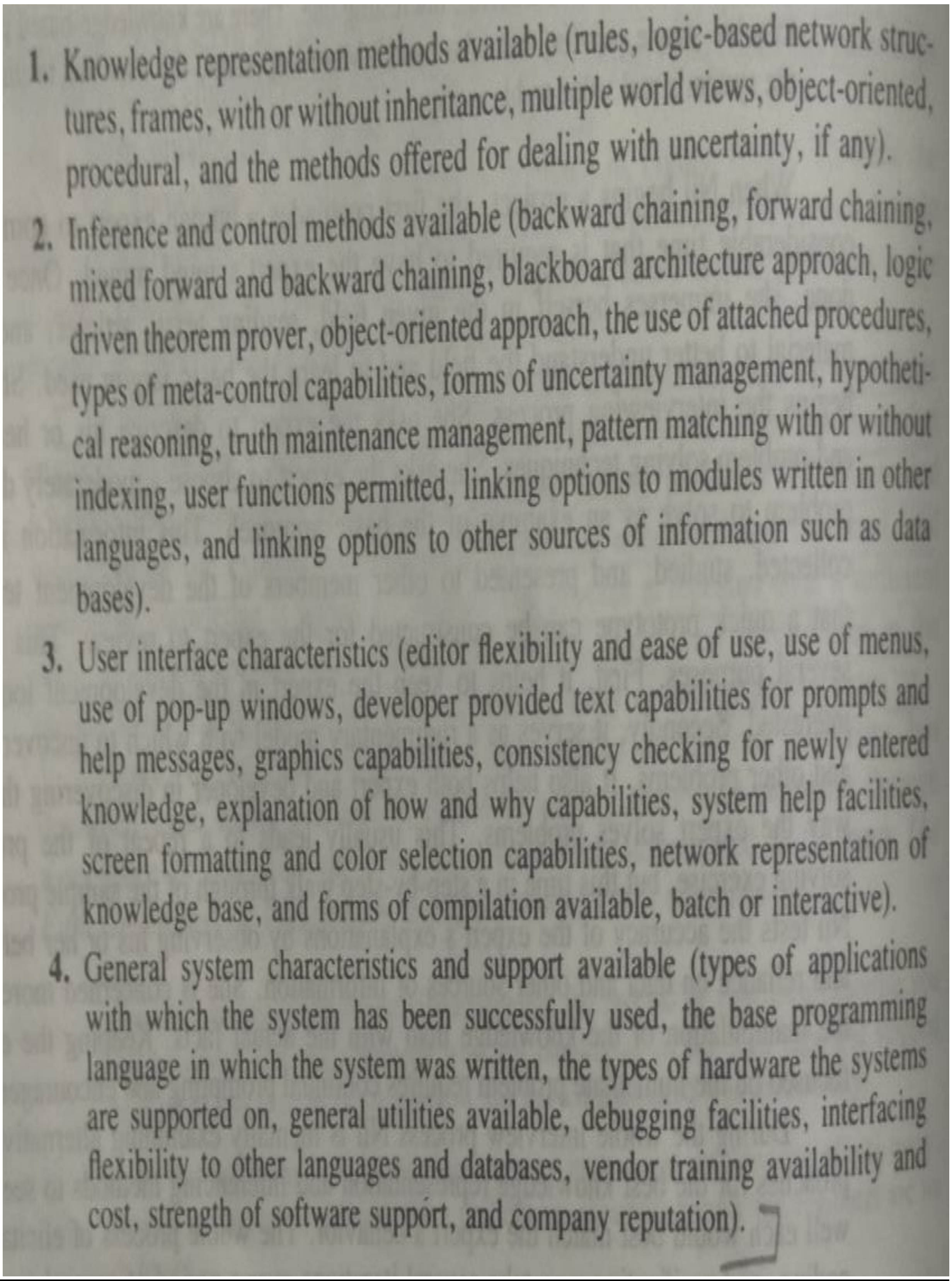


Since the introduction of the first successful expert systems in the late 1970s, a large number of building tools have been introduced, both by the academic community and industry. These tools range from high level programming languages to intelligent editors to complete shell environment systems. A number of commercial products



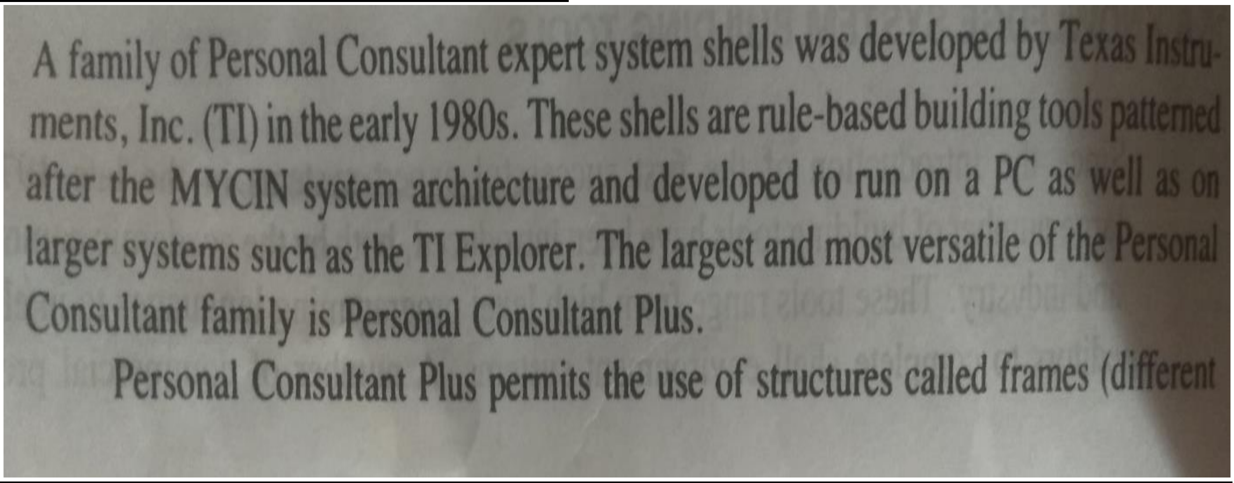
are now available ranging in price from a few hundred dollars to tens of thousands of dollars. Some are capable of running on medium size PCs while others require larger systems such as LISP machines, minis, or even main frames.

When evaluating building tools for expert system development, the developer should consider the following features and capabilities that may be offered in systems.

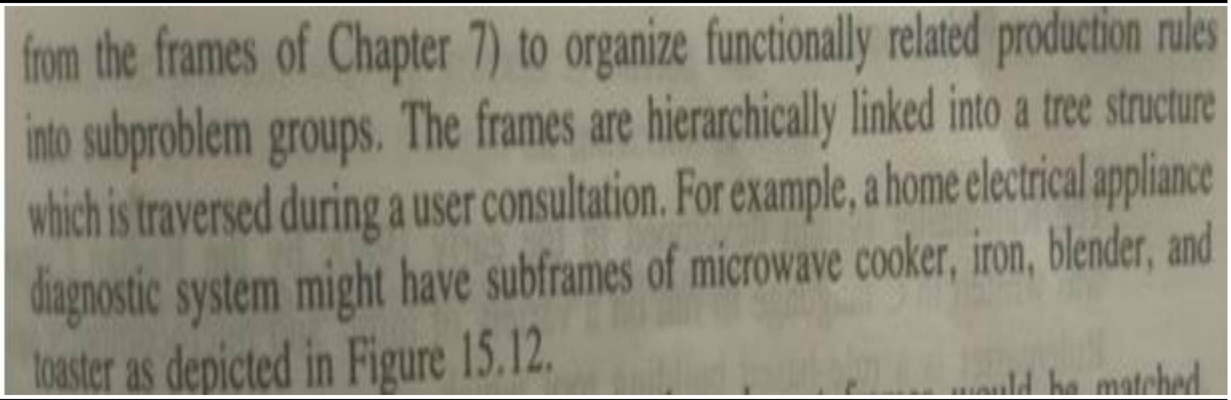


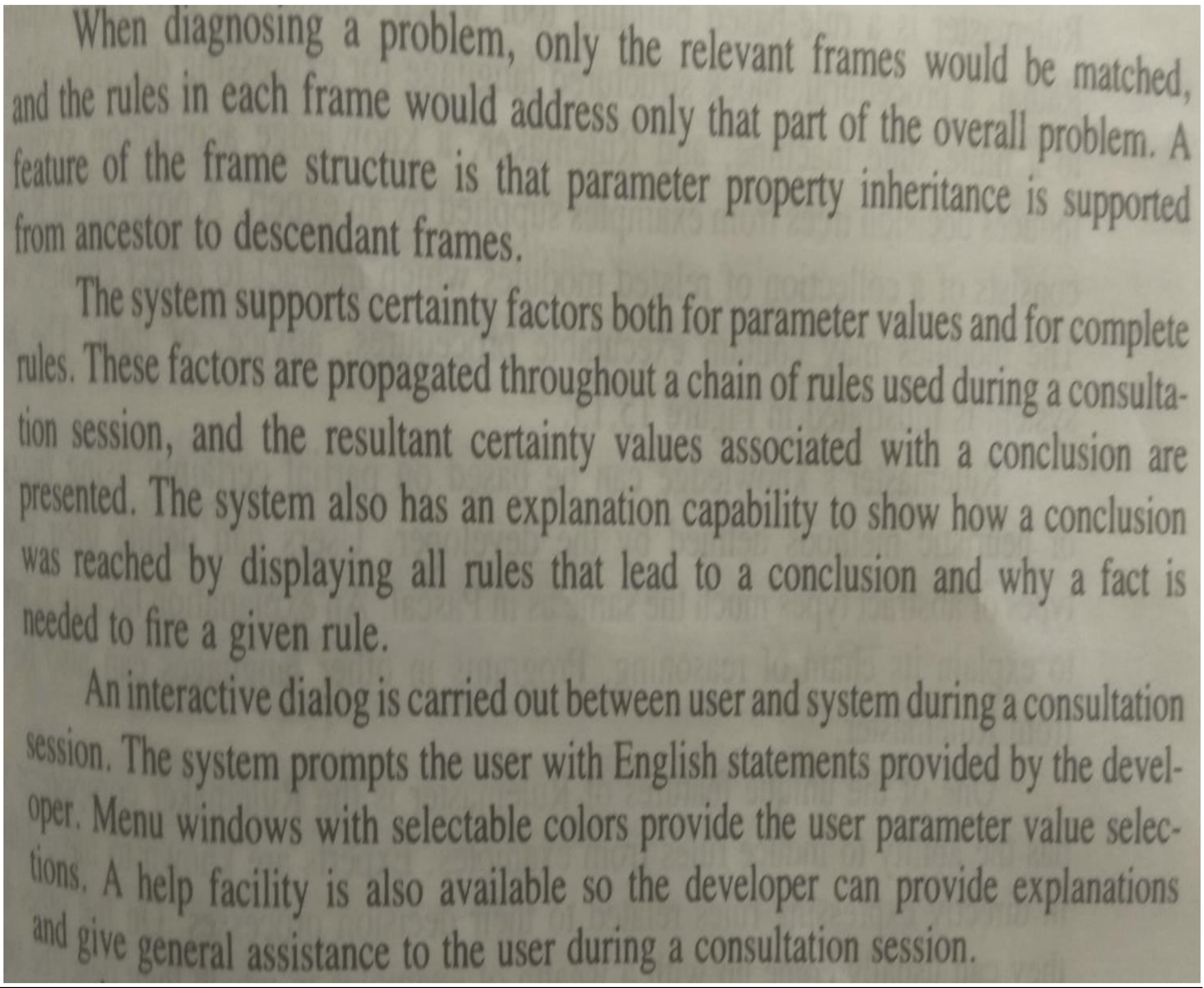
1. Knowledge representation methods available (rules, logic-based network structures, frames, with or without inheritance, multiple world views, object-oriented, procedural, and the methods offered for dealing with uncertainty, if any).
2. Inference and control methods available (backward chaining, forward chaining, mixed forward and backward chaining, blackboard architecture approach, logic driven theorem prover, object-oriented approach, the use of attached procedures, types of meta-control capabilities, forms of uncertainty management, hypothetical reasoning, truth maintenance management, pattern matching with or without indexing, user functions permitted, linking options to modules written in other languages, and linking options to other sources of information such as data bases).
3. User interface characteristics (editor flexibility and ease of use, use of menus, use of pop-up windows, developer provided text capabilities for prompts and help messages, graphics capabilities, consistency checking for newly entered knowledge, explanation of how and why capabilities, system help facilities, screen formatting and color selection capabilities, network representation of knowledge base, and forms of compilation available, batch or interactive).
4. General system characteristics and support available (types of applications with which the system has been successfully used, the base programming language in which the system was written, the types of hardware the systems are supported on, general utilities available, debugging facilities, interfacing flexibility to other languages and databases, vendor training availability and cost, strength of software support, and company reputation).

Personal Consultant Plus:


A family of Personal Consultant expert system shells was developed by Texas Instruments, Inc. (TI) in the early 1980s. These shells are rule-based building tools patterned after the MYCIN system architecture and developed to run on a PC as well as on larger systems such as the TI Explorer. The largest and most versatile of the Personal Consultant family is Personal Consultant Plus.

Personal Consultant Plus permits the use of structures called frames (different


from the frames of Chapter 7) to organize functionally related production rules into subproblem groups. The frames are hierarchically linked into a tree structure which is traversed during a user consultation. For example, a home electrical appliance diagnostic system might have subframes of microwave cooker, iron, blender, and toaster as depicted in Figure 15.12.



When diagnosing a problem, only the relevant frames would be matched, and the rules in each frame would address only that part of the overall problem. A feature of the frame structure is that parameter property inheritance is supported from ancestor to descendant frames.

The system supports certainty factors both for parameter values and for complete rules. These factors are propagated throughout a chain of rules used during a consultation session, and the resultant certainty values associated with a conclusion are presented. The system also has an explanation capability to show how a conclusion was reached by displaying all rules that lead to a conclusion and why a fact is needed to fire a given rule.

An interactive dialog is carried out between user and system during a consultation session. The system prompts the user with English statements provided by the developer. Menu windows with selectable colors provide the user parameter value selections. A help facility is also available so the developer can provide explanations and give general assistance to the user during a consultation session.

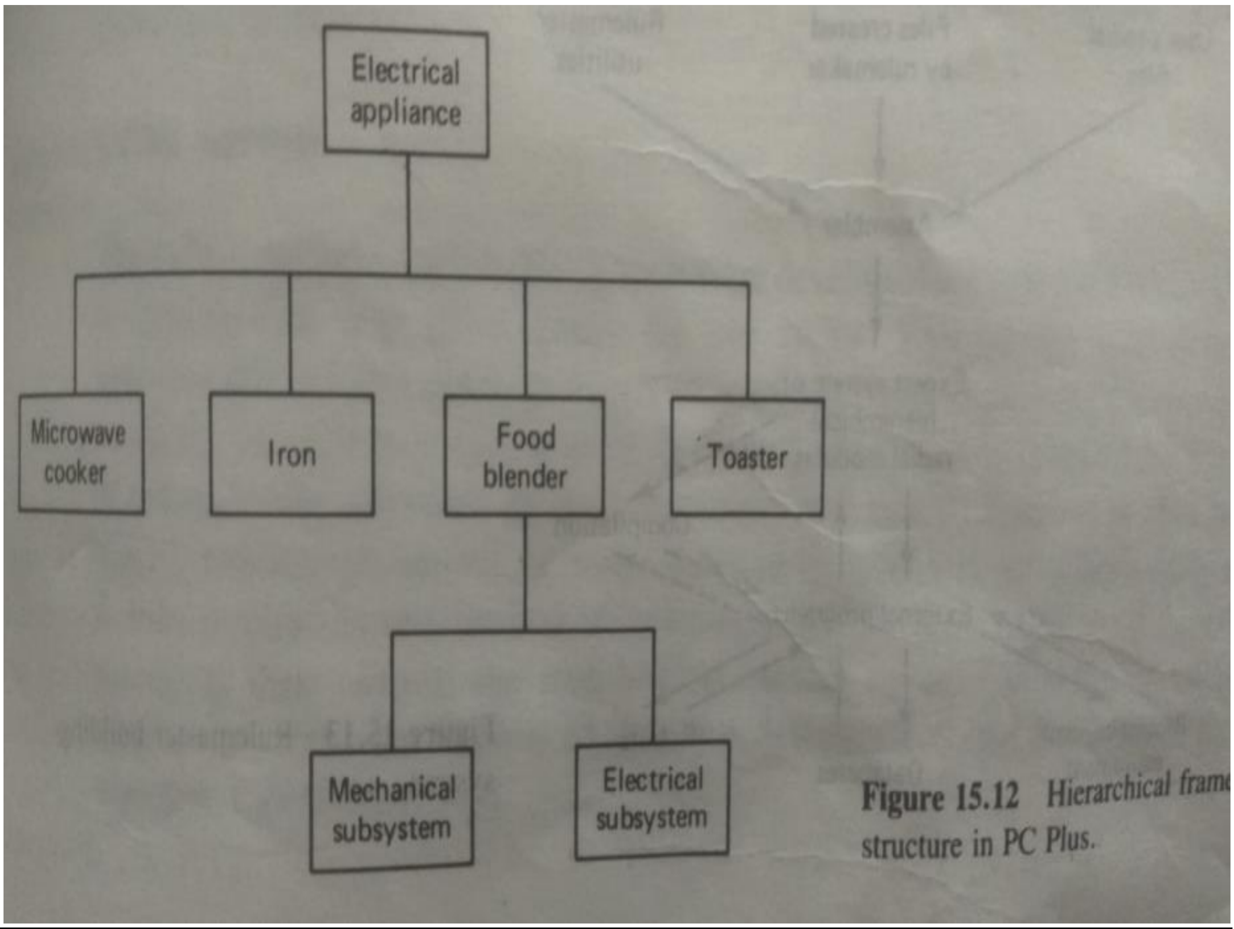
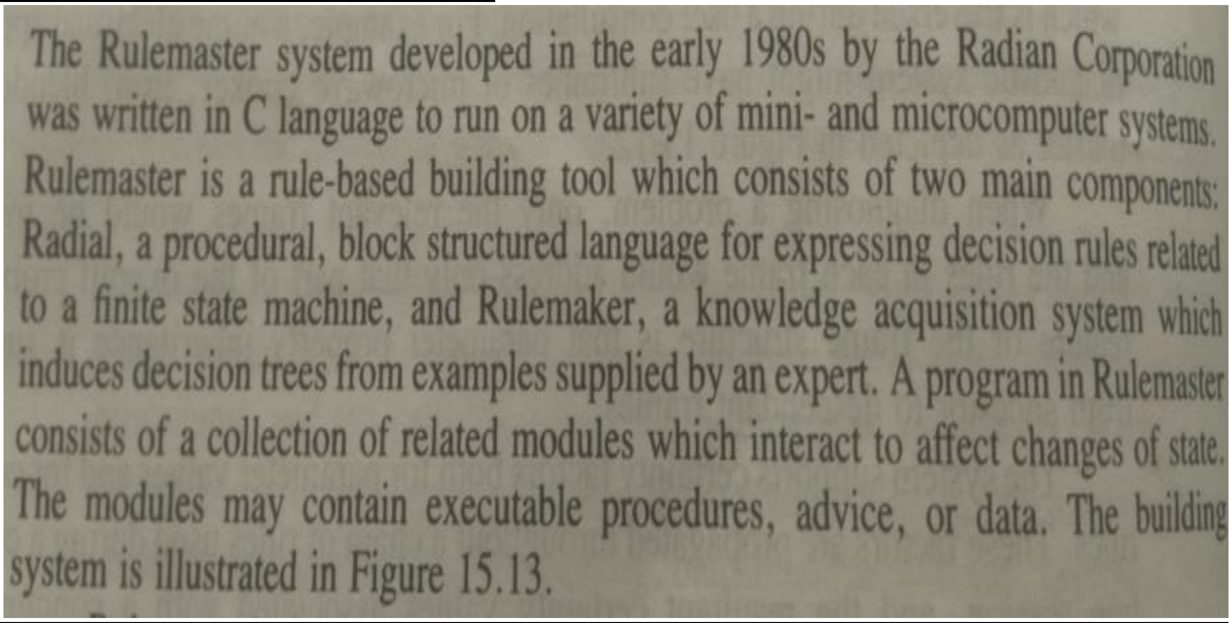
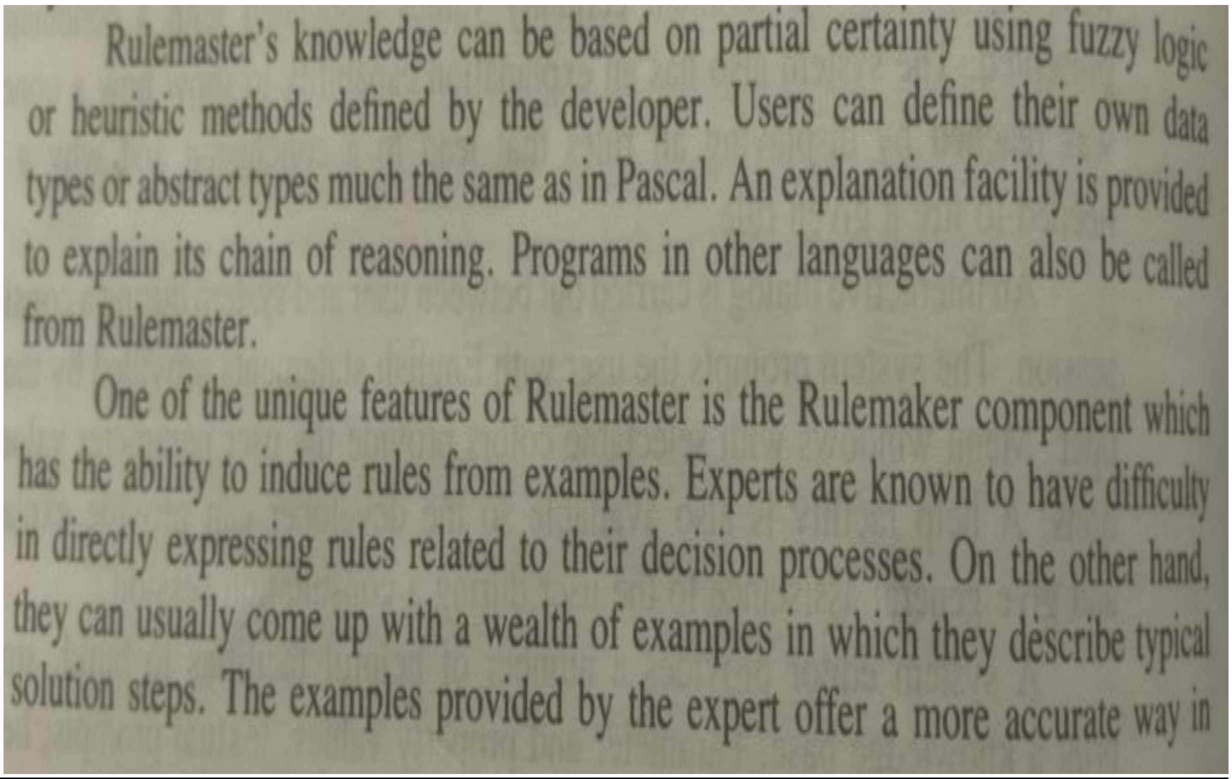


Figure 15.12 Hierarchical fram structure in PC Plus.

Radian Rulemaster:



The Rulemaster system developed in the early 1980s by the Radian Corporation was written in C language to run on a variety of mini- and microcomputer systems. Rulemaster is a rule-based building tool which consists of two main components: Radial, a procedural, block structured language for expressing decision rules related to a finite state machine, and Rulemaker, a knowledge acquisition system which induces decision trees from examples supplied by an expert. A program in Rulemaster consists of a collection of related modules which interact to affect changes of state. The modules may contain executable procedures, advice, or data. The building system is illustrated in Figure 15.13.



Rulemaster's knowledge can be based on partial certainty using fuzzy logic or heuristic methods defined by the developer. Users can define their own data types or abstract types much the same as in Pascal. An explanation facility is provided to explain its chain of reasoning. Programs in other languages can also be called from Rulemaster.

One of the unique features of Rulemaster is the Rulemaker component which has the ability to induce rules from examples. Experts are known to have difficulty in directly expressing rules related to their decision processes. On the other hand, they can usually come up with a wealth of examples in which they dèscribe typical solution steps. The examples provided by the expert offer a more accurate way in

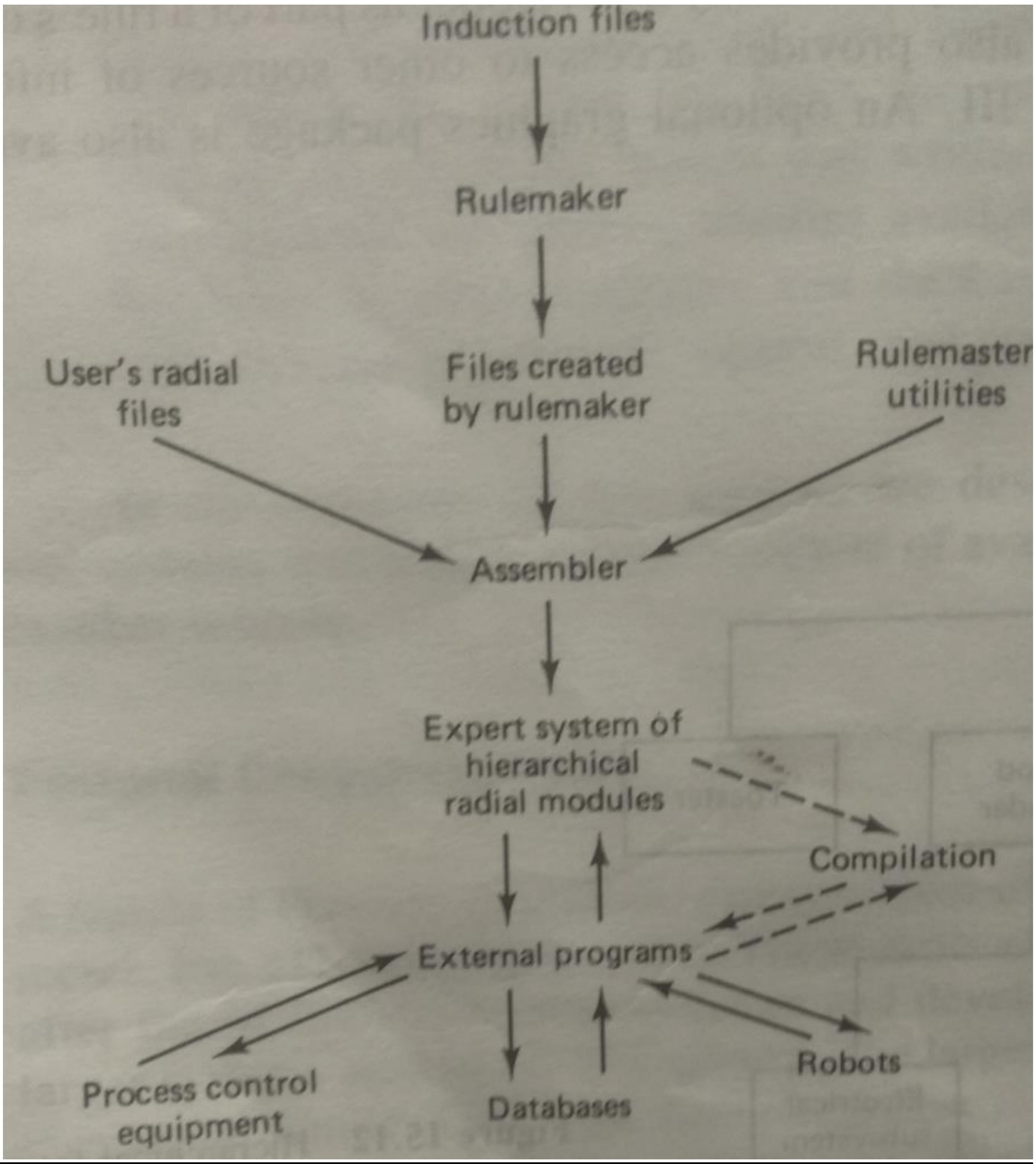


Figure 15.13. Rulemaster building system.

KEE (Knowledge Engineering Environment):

KEE (Knowledge Engineering Environment) is a frame-based development tool for Expert systems.[1] KEE was developed and sold by IntelliCorp. It was first released in 1983 and ran on Lisp Machines. KEE was later ported to Lucid Common Lisp with CLX (X11 interface for Common Lisp). This version was available on various Workstations.

On top of KEE several extensions were offered:

- Simkit,[2][3] a frame-based simulation library
- KEEconnection,[4] database connection between the frame system and relational databases.

Frames are called *Units* in KEE. Units are used for both individual instances and classes. Frames have *slots* and slots have *facets*. Facets for example describe the expected values of a slot, the inheritance rule for the slot or the value of a slot. Slots can have multiple values. Behavior can be implemented using the message-passing paradigm.

KEE provides an extensive graphical user interface to create, browse and manipulate frames.

KEE also includes a frame-based rule system. Rules themselves are frames in the KEE knowledge base. Both forward and backward chaining inference is available.

KEE supports non-monotonic reasoning through the concepts of *worlds*. Worlds allow provide alternative slot-values of frames. Through an assumption-based truth maintenance system inconsistencies can be detected and analyzed.

Active Images allows graphical displays to be attached to slots of Units. Typical examples are buttons, dials, graphs and histograms. The graphics are also implemented as Units via *KEEPictures* – a frame-based graphics library.

OPS5 SYSTEM:

OPS5 is a rule-based or production system computer language, notable as the first such language to be used in a successful expert system, the R1/XCON system used to configure VAX computers.

The OPS (said to be short for "Official Production System") family was developed in the late 1970s by Charles Forgy while at Carnegie Mellon University. Allen Newell's research group in artificial intelligence had been working on production systems for some time, but Forgy's implementation, based on his Rete algorithm, was especially efficient, sufficiently so that it was possible to scale up to larger problems involving hundreds or thousands of rules.

OPS5 uses a forward chaining inference engine; programs execute by scanning "working memory elements" (which are vaguely object-like, with classes and attributes) looking for matches with the rules in "production memory". Rules have actions that may modify or remove the matched element, create new ones, perform side effects such as output, and so forth. Execution continues until no more matches can be found.

In this sense, OPS5 is an execution engine for a Petri net extended with inhibitor arcs.

The OPS5 forward chaining process makes it extremely parallelizable during the matching phase, and several automatic parallelizing compilers were created.

OPS4 was an early version, while **OPS83** came later.

The first implementation of OPS5 was written in Lisp, and later rewritten in BLISS for speed.

DEC OPS5 is an extended implementation of the OPS5 language definition, developed for use with the VMS, RISC ULTRIX, and DEC OSF/1 operating systems.

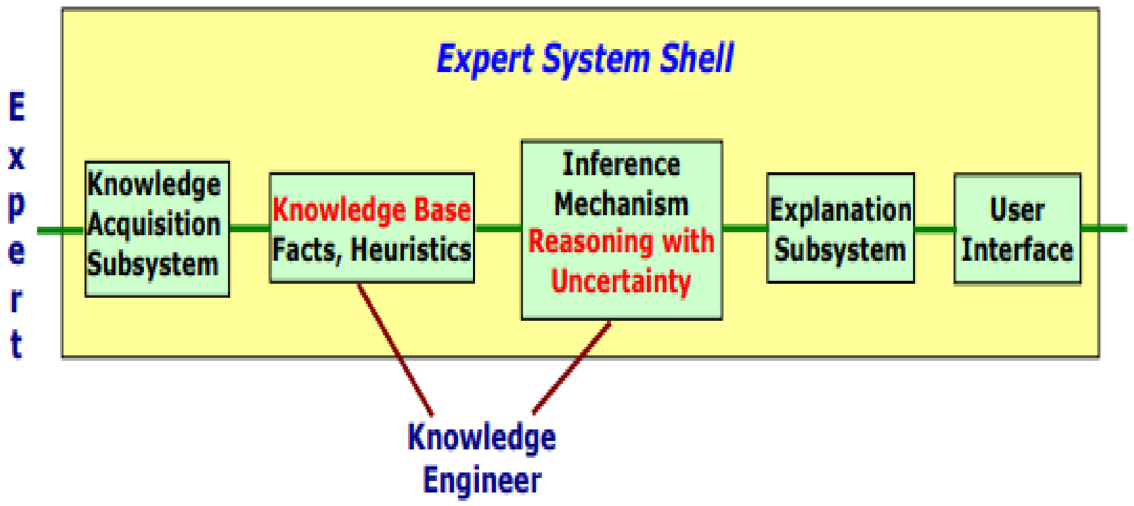
6. Expert System Shells:

Expert System Shells

An Expert system shell is a software development environment. It contains the basic components of expert systems. A shell is associated with a prescribed method for building applications by configuring and instantiating these components.

6.1 Shell components and description

The generic components of a shell : the knowledge acquisition, the knowledge Base, the reasoning, the explanation and the user interface are shown below. The knowledge base and reasoning engine are the core components.



All these components are described in the next slide.

- **Knowledge Base**

  A store of factual and heuristic knowledge. Expert system tool provides one or more knowledge representation schemes for expressing knowledge about the application domain. Some tools use both Frames (objects) and IF-THEN rules. In PROLOG the knowledge is represented as logical statements.

- **Reasoning Engine**

  Inference mechanisms for manipulating the symbolic information and knowledge in the knowledge base form a line of reasoning in solving a problem. The inference mechanism can range from simple modus ponens backward chaining of IF-THEN rules to Case-Based reasoning.

- **Knowledge Acquisition subsystem**

  A subsystem to help experts in build knowledge bases. However, collecting knowledge, needed to solve problems and build the knowledge base, is the biggest bottleneck in building expert systems.

- **Explanation subsystem**

  A subsystem that explains the system's actions. The explanation can range from how the final or intermediate solutions were arrived at justifying the need for additional data.

- **User Interface**

  A means of communication with the user. The user interface is generally not a part of the expert system technology. It was not given much attention in the past. However, the user interface can make a critical difference in the perceived utility of an Expert system.

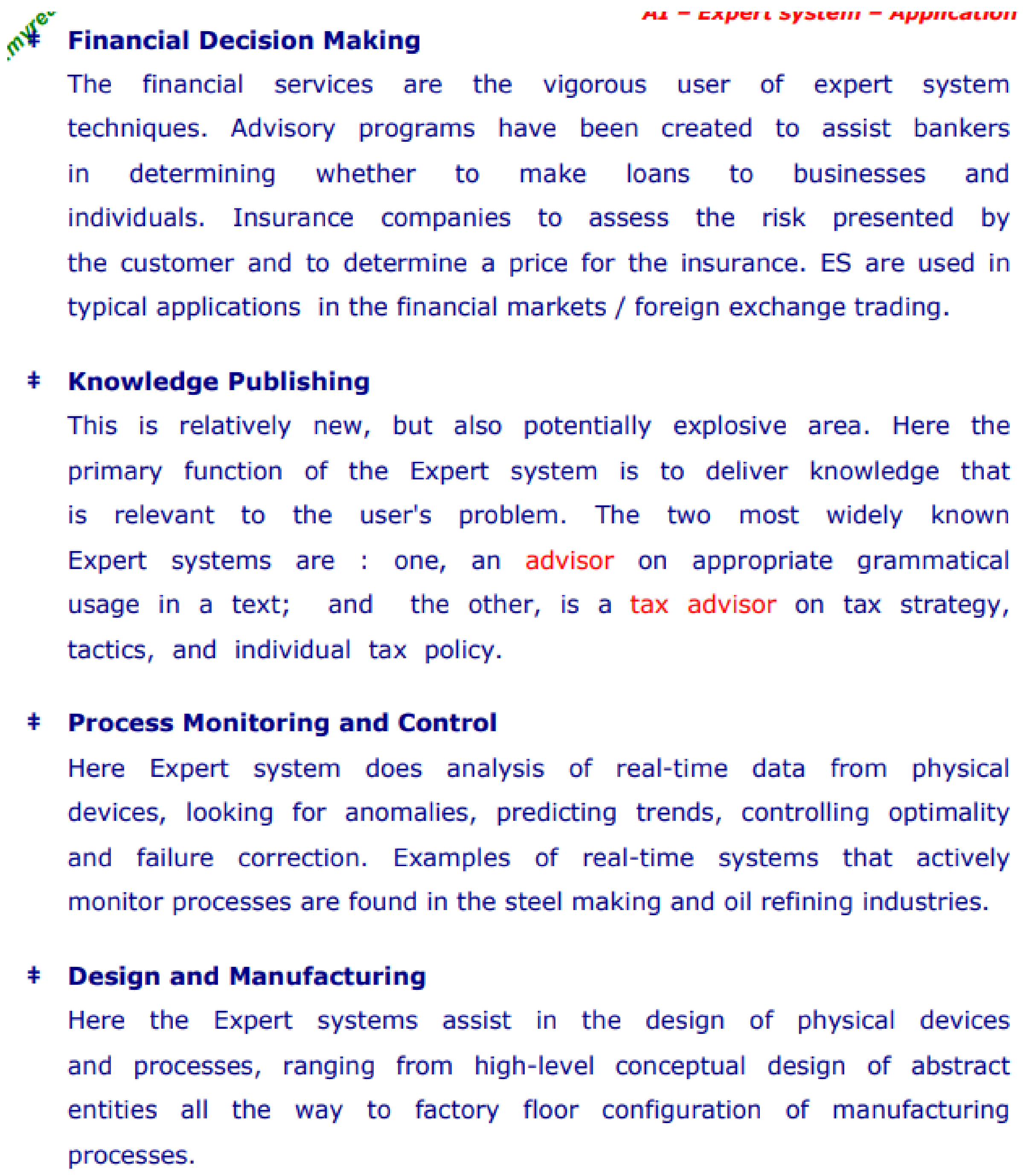
8. Application of Expert Systems

The Expert systems have found their way into most areas of knowledge work. The applications of expert systems technology have widely proliferated to industrial and commercial problems, and even helping NASA to plan the maintenance of a space shuttle for its next flight. The main applications are stated in next few slides.

- **Diagnosis and Troubleshooting of Devices and Systems**
  Medical diagnosis was one of the first knowledge areas to which Expert system technology was applied in 1976. However, the diagnosis of engineering systems quickly surpassed medical diagnosis.

- **Planning and Scheduling**
  The Expert system's commercial potential in planning and scheduling has been recognized as very large. Examples are airlines scheduling their flights, personnel, and gates; the manufacturing process planning and job scheduling;

- **Configuration of Manufactured Objects from sub-assemblies**
  Configuration problems are synthesized from a given set of elements related by a set of constraints. The Expert systems have been very useful to find solutions. For example, modular home building and manufacturing involving complex engineering design.



AI – Expert system – Application

‡ **Financial Decision Making**

The financial services are the vigorous user of expert system techniques. Advisory programs have been created to assist bankers in determining whether to make loans to businesses and individuals. Insurance companies to assess the risk presented by the customer and to determine a price for the insurance. ES are used in typical applications in the financial markets / foreign exchange trading.

‡ **Knowledge Publishing**

This is relatively new, but also potentially explosive area. Here the primary function of the Expert system is to deliver knowledge that is relevant to the user's problem. The two most widely known Expert systems are : one, an advisor on appropriate grammatical usage in a text; and the other, is a tax advisor on tax strategy, tactics, and individual tax policy.

‡ **Process Monitoring and Control**

Here Expert system does analysis of real-time data from physical devices, looking for anomalies, predicting trends, controlling optimality and failure correction. Examples of real-time systems that actively monitor processes are found in the steel making and oil refining industries.

‡ **Design and Manufacturing**

Here the Expert systems assist in the design of physical devices and processes, ranging from high-level conceptual design of abstract entities all the way to factory floor configuration of manufacturing processes.

Expert Systems: Overview of an Expert System, Architecture of an Expert Systems, Different Types of Expert Systems- Rule Based, Frame Based, and Decision Tree based, Case Based, Neural Network based, Black Board Architectures, Knowledge Acquisition and Validation Techniques, Knowledge System Building Tools, Expert System Shells.