**CHAPTER 1**

**EXPERT SYSTEMS**

**EXPERT SYSTEM**

Expert systems are computer programs that are derived from a branch of computer science research called *Artificial Intelligence* (AI). AI's scientific goal is to understand intelligence by building computer programs that exhibit intelligent behavior. It is concerned with the concepts and methods of symbolic inference, or reasoning, by a computer, and how the knowledge used to make those inferences will be represented inside the machine.

Of course, the term *intelligence* covers many cognitive skills, including the ability to solve problems, learn, and understand language; AI addresses all of those. But most progress to date in AI has been made in the area of problem solving concepts and methods for building programs that *reason* about problems rather than calculate a solution.

AI programs that achieve expert-level competence in solving problems in task areas by bringing to bear a body of knowledge about specific tasks are called *knowledge-based* or *expert systems*. Often, the term expert systems is reserved for programs whose knowledge base contains the knowledge used by human experts, in contrast to knowledge gathered from textbooks or non-experts. More often than not, the two terms, expert systems (ES) and knowledge-based systems (KBS), are used synonymously. Taken together, they represent the most widespread type of AI application. The area of human intellectual endeavor to be captured in an expert system is called the *task domain*. *Task* refers to some goal-oriented, problem-solving activity. *Domain* refers to the area within which the task is being performed. Typical tasks are diagnosis, planning, scheduling, configuration and design.

Building an expert system is known as *knowledge engineering* and its practitioners are called *knowledge engineers*. The knowledge engineer must make sure that the computer has all the knowledge needed to solve a problem. The knowledge engineer must choose one or more forms in which to represent the required knowledge as symbol patterns in the memory of the computer -- that is, he (or she) must choose a *knowledge representation*. He must also ensure that the computer can use the knowledge efficiently by selecting from a handful of *reasoning methods*. The practice of knowledge engineering is described later. We first describe the components of expert systems.

**The Building Blocks of Expert Systems**

Every expert system consists of two principal parts: the knowledge base; and the reasoning, or inference, engine.

The *knowledge base* of expert systems contains both factual and heuristic knowledge. *Factual knowledge* is that knowledge of the task domain that is widely shared, typically found in textbooks or journals, and commonly agreed upon by those knowledgeable in the particular field.

*Heuristic knowledge* is the less rigorous, more experiential, more judgmental knowledge of performance. In contrast to factual knowledge, heuristic knowledge is rarely discussed, and is largely individualistic. It is the knowledge of good practice, good judgment, and plausible reasoning in the field. It is the knowledge that underlies the "art of good guessing."

*Knowledge representation* formalizes and organizes the knowledge. One widely used representation is the *production rule*, or simply *rule*. A rule consists of an IF part and a THEN part (also called a *condition* and an *action*). The IF part lists a set of conditions in some logical combination. The piece of knowledge represented by the production rule is relevant to the line of reasoning being developed if the IF part of the rule is satisfied; consequently, the THEN part can be concluded, or its problem-solving action taken. Expert systems whose knowledge is represented in rule form are called *rule-based systems*.

Another widely used representation, called the *unit* (also known as *frame*, *schema*, or *list structure*) is based upon a more passive view of knowledge. The unit is an assemblage of associated symbolic knowledge about an entity to be represented. Typically, a unit consists of a list of properties of the entity and associated values for those properties.

Since every task domain consists of many entities that stand in various relations, the properties can also be used to specify relations, and the values of these properties are the names of other units that are linked according to the relations. One unit can also represent knowledge that is a "special case" of another unit, or some units can be "parts of" another unit.

The *problem-solving model,* or *paradigm*, organizes and controls the steps taken to solve the problem. One common but powerful paradigm involves chaining of IF-THEN rules to form a line of reasoning. If the chaining starts from a set of conditions and moves toward some conclusion, the method is called *forward chaining*. If the conclusion is known (for example, a goal to be achieved) but the path to that conclusion is not known, then reasoning backwards is called for, and the method is *backward chaining*. These problem-solving methods are built into program modules called *inference engines* or *inference procedures* that manipulate and use knowledge in the knowledge base to form a line of reasoning.

The *knowledge base* an expert uses is what he learned at school, from colleagues, and from years of experience. Presumably the more experience he has, the larger his store of knowledge. Knowledge allows him to interpret the information in his databases to advantage in diagnosis, design, and analysis.

Though an expert system consists primarily of a knowledge base and an inference engine, a couple of other features are worth mentioning: reasoning with uncertainty, and explanation of the line of reasoning.

Knowledge is almost always incomplete and uncertain. To deal with uncertain knowledge, a rule may have associated with it a *confidence factor* or a weight. The set of methods for using uncertain knowledge in combination with uncertain data in the reasoning process is called *reasoning with uncertainty*. An important subclass of methods for reasoning with uncertainty is called "fuzzy logic," and the systems that use them are known as "fuzzy systems."

Because an expert system uses uncertain or heuristic knowledge (as we humans do) its credibility is often in question (as is the case with humans). When an answer to a problem is questionable, we tend to want to know the rationale. If the rationale seems plausible, we tend to believe the answer. So it is with expert systems. Most expert systems have the ability to answer questions of the form: "Why is the answer X?" Explanations can be generated by tracing the line of reasoning used by the inference engine (Feigenbaum, McCorduck et al. 1988).

The most important ingredient in any expert system is knowledge. The power of expert systems resides in the specific, high-quality knowledge they contain about task domains. AI researchers will continue to explore and add to the current repertoire of knowledge representation and reasoning methods. But in knowledge resides the power. Because of the importance of knowledge in expert systems and because the current knowledge acquisition method is slow and tedious, much of the future of expert systems depends on breaking the knowledge acquisition bottleneck and in codifying and representing a large knowledge infrastructure.

**Knowledge engineering**

is the art of designing and building expert systems, and knowledge engineers are its practitioners. Gerald M. Weinberg said of programming in *The Psychology of Programming*: "'Programming,' -- like 'loving,' -- is a single word that encompasses an infinitude of activities" (Weinberg 1971). Knowledge engineering is the same, perhaps more so. We stated earlier that knowledge engineering is an applied part of the science of artificial intelligence which, in turn, is a part of computer science. Theoretically, then, a knowledge engineer is a computer scientist who knows how to design and implement programs that incorporate artificial intelligence techniques. The nature of knowledge engineering is changing, however, and a new breed of knowledge engineers is emerging. We'll discuss the evolving nature of knowledge engineering later.

Today there are two ways to build an expert system. They can be built from scratch, or built using a piece of development software known as a "tool" or a "shell." Before we discuss these tools, let's briefly discuss what knowledge engineers do. Though different styles and methods of knowledge engineering exist, the basic approach is the same: a knowledge engineer interviews and observes a human expert or a group of experts and learns what the experts know, and how they reason with their knowledge. The engineer then translates the knowledge into a computer-usable language, and designs an inference engine, a reasoning structure, that uses the knowledge appropriately. He also determines how to integrate the use of uncertain knowledge in the reasoning process, and what kinds of explanation would be useful to the end user.

Next, the inference engine and facilities for representing knowledge and for explaining are programmed, and the domain knowledge is entered into the program piece by piece. It may be that the inference engine is not just right; the form of knowledge representation is awkward for the kind of knowledge needed for the task; and the expert might decide the pieces of knowledge are wrong. All these are discovered and modified as the expert system gradually gains competence.

The discovery and accumulation of techniques of machine reasoning and knowledge representation is generally the work of artificial intelligence research. The discovery and cumulation of knowledge of a task domain is the province of domain experts. Domain knowledge consists of both formal, textbook knowledge, and experiential knowledge -- the *expertise* of the experts.

**Tools, Shells, and Skeletons**

Compared to the wide variation in domain knowledge, only a small number of AI methods are known that are useful in expert systems. That is, currently there are only a handful of ways in which to represent knowledge, or to make inferences, or to generate explanations. Thus, systems can be built that contain these useful methods without any domain-specific knowledge. Such systems are known as *skeletal systems, shells*, or simply *AI tools*.

Building expert systems by using shells offers significant advantages. A system can be built to perform a unique task by entering into a shell all the necessary knowledge about a task domain. The inference engine that applies the knowledge to the task at hand is built into the shell. If the program is not very complicated and if an expert has had some training in the use of a shell, the expert can enter the knowledge himself.

Many commercial shells are available today, ranging in size from shells on PCs, to shells on workstations, to shells on large mainframe computers. They range in price from hundreds to tens of thousands of dollars, and range in complexity from simple, forward-chained, rule-based systems requiring two days of training to those so complex that only highly trained knowledge engineers can use them to advantage. They range from general-purpose shells to shells custom-tailored to a class of tasks, such as financial planning or real-time process control.

Although shells simplify programming, in general they don't help with knowledge acquisition. *Knowledge acquisition* refers to the task of endowing expert systems with knowledge, a task currently performed by knowledge engineers. The choice of reasoning method, or a shell, is important, but it isn't as important as the accumulation of high-quality knowledge. The power of an expert system lies in its store of knowledge about the task domain -- the more knowledge a system is given, the more competent it becomes.

**Bricks and Mortar**

The fundamental working hypothesis of AI is that intelligent behavior can be precisely described as symbol manipulation and can be modeled with the symbol processing capabilities of the computer.

In the late 1950s, special programming languages were invented that facilitate symbol manipulation. The most prominent is called LISP (LISt Processing). Because of its simple elegance and flexibility, most AI research programs are written in LISP, but commercial applications have moved away from LISP.

In the early 1970s another AI programming language was invented in France. It is called PROLOG (PROgramming in LOGic). LISP has its roots in one area of mathematics (lambda calculus), PROLOG in another (first-order predicate calculus).

PROLOG consists of English-like statements which are facts (assertions), rules (of inference), and questions. Here is an inference rule: "If object-x is part-of object-y then a component-of object-y is object-x."

Programs written in PROLOG have behavior similar to rule-based systems written in LISP. PROLOG, however, did not immediately become a language of choice for AI programmers. In the early 1980s it was given impetus with the announcement by the Japanese that they would use a logic programming language for the Fifth Generation Computing Systems (FGCS) Project. A variety of logic-based programming languages have since arisen, and the term *prolog* has become generic.

**CHARACTERISTICS OF AN EXPERT SYSTEM**

 The growth of expert system is expected to continue for several years. With the continuing growth, many new and exciting applications will emerge. An expert system operates as an interactive system that responds to questions, asks for clarification, makes recommendations and generally aids the decision making process. Expert system provides expert advice and guidance in a wide variety of activities from computer diagnosis to delicate medical surgery.

An expert system is usually designed to have the following general characteristics.

1.     **High level Performance:**The system must be capable of responding at a level of competency equalto or better than an expert system in the field. The quality of the advice given by the system should be in a high level integrity and for which the performance ratio should be also very high.

2.     **Domain Specificity:**Expert systems are typically very domain specific. For ex., a diagnostic expertsystem for troubleshooting computers must actually perform all the necessary data manipulation as a human expert would. The developer of such a system must limit his or her scope of the system to just what is needed to solve the target problem. Special tools or programming languages are often needed to accomplish the specific objectives of the system.

3.     **Good Reliability:**The expert system must be as reliable as a human expert.

4.     **Understandable:**The system should be understandable i.e. be able to explain the steps of reasoningwhile executing. The expert system should have an explanation capability similar to the reasoning ability of human experts.

5.     **Adequate Response time:**The system should be designed in such a way that it is able to performwithin a small amount of time, comparable to or better than the time taken by a human expert to reach at a decision point. An expert system that takes a year to reach a decision compared to a human expert’s time of one hour would not be useful.

6.     **Use symbolic representations:**Expert system use symbolic representations for knowledge (rules,networks or frames) and perform their inference through symbolic computations that closely resemble manipulations of natural language.

7.     **Linked with Metaknowledge:**Expert systems often reason with metaknowledge i.e. they reasonwith knowledge about themselves and their own knowledge limits and capabilities. The use of metaknowledge is quite interactive and simple for various data representations.

8.     **Expertise knowledge:**Real experts not only produce good solutions but also find them quickly. So,an expert system must be skillful in applying its knowledge to produce solutions both efficiently and effectively by using the intelligence human experts.

9.     **Justified Reasoning:**This allows the users to ask the expert system to justify the solution or adviceprovided by it. Normally, expert systems justify their answers or advice by explaining their reasoning. If a system is a rule based system, it provides to the user all the rules and facts it has used to achieve its answer.

10.            **Explaining capability:**Expert systems are capable of explaining how a particular conclusion wasreached and why requested information is needed during a consultation. This is very important as it gives the user a chance to access and understand the system’s reasoning ability, thereby improving the user’s confidence in the system.

11.            **Special Programming Languages:**Expert systems are typically written in special programminglanguages. The use of languages like LISP and PROLOG in the development of an expert system simplifies the coding process. The major advantage of these languages, as compared to conventional programming languages is the simplicity of the addition, elimination or substitution of new rules and memory management capabilities. Some of the distinguishing characteristics of programming languages needed for expert system work are as follows:

            Efficient mix of integer and real variables.

            Good memory management procedures.

            Extensive data manipulation routines.

            Incremental compilation.

            Tagged memory architecture.

            Efficient search procedures.

            Optimization of the systems environment.

**The Need for Expert System**

Expert systems can automate specific knowledge based tasks such as credit approval, fraud detection, loan application scoring, and so forth. Knowledge from a human expert can be translated into rules or cases and applied to datasets collected from information gathering systems. Information jobs relying upon human expertise for decision making are the best candidates to be automated by expert systems. The cost of maintaining expert systems may be high due to the level of conflicting rules within a given system. Standard cost benefit trade off analysis is typically performed before an expert system project is undertaken.

Expert systems were intended to capture the knowledge of experts in a question-based format such that others, including computers, could apply that expertise more broadly. Ideally this could be applied to a wide variety of problems such as medical diagnosis.

This approach was successful in problems of limited scope, but more complex problems turned out to be unwieldy in terms of amount of information to capture and time needed to interview the experts. Additionally, many times the experts’ knowledge was intuition-based and could not be captured in a formal manner.

**Goals, roles and advantages/benefits of Expert System**

**Objectives of an Expert System**

* Capturing human expertise.

Why?

A catalogue of reasons:

- share (with colleagues, i.e., other experts; like consultation)

- spread to areas where unavailable (maybe only one expert in this area exists)

- perpetuate (what if experts leaves your employ? or dies?)

- obviate expert's role (related to all of the above, especially "spread")

- clarify (gives insights into problem domain)

But the most significant reason might be:

- the ideas and techniques are extensible to other areas

That is,

- expert systems themselves are useful in learning about or developing other topics

- but, as is more often the case, techniques and pedagogies used in studying expert systems are useful in learning about other areas

Ancillary Objectives of the Study of Expert Systems

The capturing of expertise allows exposition of other possibilities:

* Understanding of general principles of expertise
* Categorization of domain specific principles of expertise
* Possible improvement of performance by refinement of these principles (improvement in both the system and individual experts from which the system was derived)

**Roles in Expert System Development**

* Three fundamental roles in building expert systems are:
* 1. ***Expert*** - Successful ES systems depend on the experience and application of knowledge that the people can bring to it during its development. Large systems generally require multiple experts.
* 2. ***Knowledge engineer -***The knowledge engineer has a dual task. This person should be able to elicit knowledge from the expert, gradually gaining an understanding of an area of expertise. Intelligence, tact, empathy, and proficiency in specific techniques of knowledge acquisition are all required of a knowledge engineer. Knowledge-acquisition techniques include conducting interviews with varying degrees of structure, protocol analysis, observation of experts at work, and analysis of cases.
* On the other hand, the knowledge engineer must also select a tool appropriate for the project and use it to represent the knowledge with the application of the ***knowledge acquisition facility***.
* 3. ***User*** - A system developed by an end user with a simple shell, is built rather quickly an inexpensively. Larger systems are built in an organized development effort. A prototype-oriented iterative development strategy is commonly used. ESs lends themselves particularly well to prototyping.

### Advantages of Using Expert System:

An expert system has been reliably used in the business world to gain tactical advantages and forecast the market’s condition. In this globalization era where every decision made in the business world is critical for success, the assistance provided from an expert system is undoubtedly essential and highly reliable for an organization to succeed.

Examples given below will be the advantages for the implementation of an expert system in business:

* Providing consistent solutions: It can provide consistent answers for repetitive decisions, processes, and tasks. As long as the rule base in the system remains the same, regardless of how many times similar problems are being tested, the final conclusions drawn will remain the same.
* Provides reasonable explanations: It has the ability to clarify the reasons why the conclusion was drawn and be why it is considered as the most logical choice among other alternatives. If there are any doubts in concluding a certain problem, it will prompt some questions for users to answer in order to process the logical conclusion.
* Overcome human limitations: It does not have human limitations and can work around the clock continuously. Users will be able to frequently use it in seeking solutions. The knowledge of experts is an invaluable asset for the company. It can store the knowledge and use it as long as the organization needs.
* Easy to adapt to new conditions: Unlike humans who often have troubles in adapting in new environments, an expert system has high adaptability and can meet new requirements in a short period of time. It also can capture new knowledge from an expert and use it as inference rules to solve new problems.

## Common Application Areas

**Applications of Expert Systems**

The test outlines some illustrative minicases of expert systems applications. These include areas such as high-risk credit decisions, advertising decision making, and manufacturing decisions.

**Generic Categories of Expert System Applications**

Application areas include classification, diagnosis, monitoring, process control, design, scheduling and planning, and generation of options.

*Classification* - identify an object based on stated characteristics

*Diagnosis Systems* - infer malfunction or disease from observable data

*Monitoring* - compare data from a continually observed system to prescribe behaviour

*Process Control* - control a physical process based on monitoring

*Design* - configure a system according to specifications

*Scheduling & Planning* - develop or modify a plan of action

*Generation of Options* - generate alternative solutions to a problem

## THE APPLICATIONS OF EXPERT SYSTEMS

The spectrum of applications of expert systems technology to industrial and commercial problems is so wide as to defy easy characterization. The applications find their way into most areas of knowledge work. They are as varied as helping salespersons sell modular factory-built homes to helping NASA plan the maintenance of a space shuttle in preparation for its next flight.

Applications tend to cluster into seven major classes.

### Diagnosis and Troubleshooting of Devices and Systems of All Kinds

This class comprises systems that deduce faults and suggest corrective actions for a malfunctioning device or process. Medical diagnosis was one of the first knowledge areas to which ES technology was applied (for example, see Shortliffe 1976), but diagnosis of engineered systems quickly surpassed medical diagnosis. There are probably more diagnostic applications of ES than any other type. The diagnostic problem can be stated in the abstract as: given the evidence presenting itself, what is the underlying problem/reason/cause?

### Planning and Scheduling

Systems that fall into this class analyze a set of one or more potentially complex and interacting goals in order to determine a set of actions to achieve those goals, and/or provide a detailed temporal ordering of those actions, taking into account personnel, materiel, and other constraints. This class has great commercial potential, which has been recognized. Examples involve airline scheduling of flights, personnel, and gates; manufacturing job-shop scheduling; and manufacturing process planning.

### Configuration of Manufactured Objects from Subassemblies

Configuration, whereby a solution to a problem is synthesized from a given set of elements related by a set of constraints, is historically one of the most important of expert system applications. Configuration applications were pioneered by computer companies as a means of facilitating the manufacture of semi-custom minicomputers (McDermott 1981). The technique has found its way into use in many different industries, for example, modular home building, manufacturing, and other problems involving complex engineering design and manufacturing.

### Financial Decision Making

The financial services industry has been a 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 have used expert systems to assess the risk presented by the customer and to determine a price for the insurance. A typical application in the financial markets is in foreign exchange trading.

### Knowledge Publishing

This is a relatively new, but also potentially explosive area. The primary function of the expert system is to deliver knowledge that is relevant to the user's problem, in the context of the user's problem. The two most widely distributed expert systems in the world are in this category. The first is an advisor which counsels a user on appropriate grammatical usage in a text. The second is a tax advisor that accompanies a tax preparation program and advises the user on tax strategy, tactics, and individual tax policy.

### Process Monitoring and Control

Systems falling in this class analyze real-time data from physical devices with the goal of noticing anomalies, predicting trends, and controlling for both optimality and failure correction. Examples of real-time systems that actively monitor processes can be found in the steel making and oil refining industries.

### Design and Manufacturing

These 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.

## CHAPTER 2

## EXPERT SYSTEM COMPONENET FACTS

## STRUCTURE OF EXPERT SYSTEMS

One of the largest area of applications of artificial intelligence is in expert sytems, or knowledge based systems as they are often known. This type of system seeks to exploit the specialised skills or information held by of a group of people on specific areas. It can be thought of as a computerised consulting service. It can also be called an information guidance system. Such systems are used for prospecting medical diagnosis or as educational aids. They are also used in engineering and manufacture in the control of robots where they inter-relate with vision systems. The initial attempts to apply artificial intelligence to generalised problems made limited progress as we have seen but it was soon realised that more significant progress could be made if the field of interest was restricted.

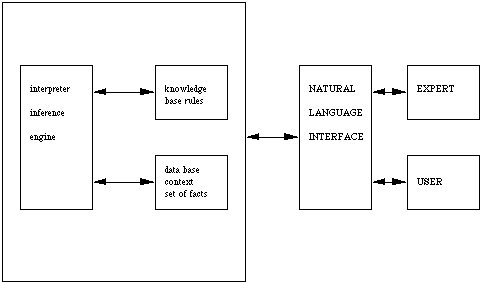
STRUCTURE

The internal structure of an expert system can be considered to consist of three parts:

the knowledge base ; the database; the rule interpreter.

This is analagous to the production system where we have

the set of productions; the set of facts held as working memory and a rule interpreter.



The knowledge base holds the set of rules of inference that are used in reasoning. Most of these systems use IF-THEN rules to represent knowledge. Typically systems can have from a few hundred to a few thousand rules.

The database gives the context of the problem domain and is generally considered to be a set of useful facts. These are the facts that satisfy the condition part of the condition action rules as the IF THEN rules can be thought of.

The rule interpreter is often known as an inference engine and controls the knowledge base using the set of facts to produce even more facts. Communication with the sytem is ideally provided by a natural language interface. This enables a user to interact indelendently of the expert with the intelligent system.

OPERATION OF THE SYSTEM

Again there are three modes to this:

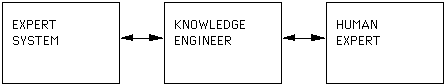
the knowledge acquisition mode;

the consultation mode;

and the explanation mode.

We shall consider each in turn.

KNOWLEDGE ACQUISITION



The system must liaise with people in order to gain knowledge and the people must be specialised in the appropriate area of activity. For example medical doctors, geologists or chemists. The knowledge engineer acts as an intermediary between the specialist and the expert system. Typical of the information that must be gleaned is

vocabulary or jargon, general concepts and facts, problems that commonly arise, the solutions to the problems that occur and skills for solving particular problems. This process of picking the brain of an expert is a specialised form of data capture and makes use of interview techniques. The knowledge engineer is also responsible for the self consistency of the data loaded. Thus a number of specific tests have to be performed to ensure that the conclusions reached are sensible.

User Interface

User interface provides interaction between user of the ES and the ES itself. It is generally Natural Language Processing so as to be used by the user who is well-versed in the task domain. The user of the ES need not be necessarily an expert in Artificial Intelligence.

It explains how the ES has arrived at a particular recommendation. The explanation may appear in the following forms −

* Natural language displayed on screen.
* Verbal narrations in natural language.
* Listing of rule numbers displayed on the screen.

The user interface makes it easy to trace the credibility of the deductions.

Requirements of Efficient ES User Interface

* It should help users to accomplish their goals in shortest possible way.
* It should be designed to work for user’s existing or desired work practices.
* Its technology should be adaptable to user’s requirements; not the other way round.
* It should make efficient use of user input.

Inference Engine

Use of efficient procedures and rules by the Inference Engine is essential in deducting a correct, flawless solution.

In case of knowledge-based ES, the Inference Engine acquires and manipulates the knowledge from the knowledge base to arrive at a particular solution.

In case of rule based ES, it −

* Applies rules repeatedly to the facts, which are obtained from earlier rule application.
* Adds new knowledge into the knowledge base if required.
* Resolves rules conflict when multiple rules are applicable to a particular case.

To recommend a solution, the Inference Engine uses the following strategies −

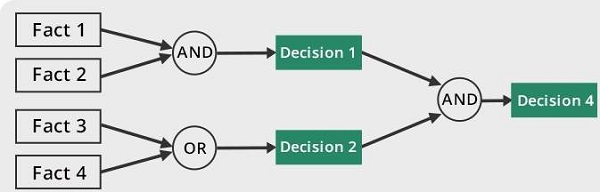
* Forward Chaining
* Backward Chaining

Forward Chaining

It is a strategy of an expert system to answer the question, **“What can happen next?”**

Here, the Inference Engine follows the chain of conditions and derivations and finally deduces the outcome. It considers all the facts and rules, and sorts them before concluding to a solution.

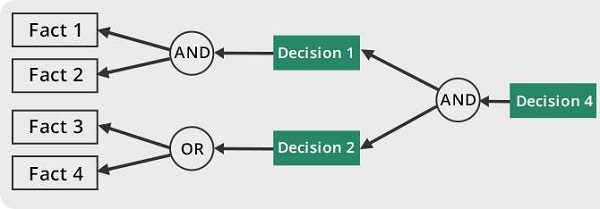
This strategy is followed for working on conclusion, result, or effect. For example, prediction of share market status as an effect of changes in interest rates.



Backward Chaining

With this strategy, an expert system finds out the answer to the question, **“Why this happened?”**

On the basis of what has already happened, the Inference Engine tries to find out which conditions could have happened in the past for this result. This strategy is followed for finding out cause or reason. For example, diagnosis of blood cancer in humans.



## Knowledge Base

It contains domain-specific and high-quality knowledge.

Knowledge is required to exhibit intelligence. The success of any ES majorly depends upon the collection of highly accurate and precise knowledge.

### What is Knowledge?

The data is collection of facts. The information is organized as data and facts about the task domain. **Data, information,** and **past experience** combined together are termed as knowledge.

### Components of Knowledge Base

The knowledge base of an ES is a store of both, factual and heuristic knowledge.

* **Factual Knowledge** − It is the information widely accepted by the Knowledge Engineers and scholars in the task domain.
* **Heuristic Knowledge** − It is about practice, accurate judgement, one’s ability of evaluation, and guessing.

### Knowledge representation

It is the method used to organize and formalize the knowledge in the knowledge base. It is in the form of IF-THEN-ELSE rules.

### Knowledge Acquisition

The success of any expert system majorly depends on the quality, completeness, and accuracy of the information stored in the knowledge base.

The knowledge base is formed by readings from various experts, scholars, and the **Knowledge Engineers**. The knowledge engineer is a person with the qualities of empathy, quick learning, and case analyzing skills.

He acquires information from subject expert by recording, interviewing, and observing him at work, etc. He then categorizes and organizes the information in a meaningful way, in the form of IF-THEN-ELSE rules, to be used by interference machine. The knowledge engineer also monitors the development of the ES.

Knowledge Engineering

Definition: knowledge engineering - the process of building an exp. system.

K.E. essentially includes

1. acquiring knowledge from human expert (or other source)

and

1. encoding it into a system.

Steps in the knowledge engineering process (cf., waterfall and spiral methods of software design)

Phase 1

<-------- reformulation ------------------

Assessment |

| |

Requirements | |

V |

Phase 2 |

<-- Explorations ------ |

Knowledge Acquisition | |

| | |

Knowledge | | |

V | |

Phase 3 | |

<- refinements ------ | |

Design | | |

| | | |

Structure | | | |

V | | |

Phase 4 | | |

------------------------------------- |

Test |

| |

Evaluation | |

V |

Phase 5 |

|

Documentation |

| |

Product | |

V |

Phase 6 |

-----------------------------------------

Maintenance

**CHAPTER 3**

**KNOWLEDGE REPRESENTATION**

**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.

**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.

Knowledge Representation Techniques

There is an unknowable number of ways to represent knowledge. Most that are easily conceivable to human-beings are constructed from a small subset of possibilities.

* Semantic Networks
* Frames
* Logic

Semantic Networks

The most *general* representation, hence usually the most difficult to easily implement correctly.

Note - most speed-up comes from *specialization*. The trick is to move toward more generality *and* maintain ease-of-use, speed, etc.

Advantages of generality

* key points of the ``problem class'' are identified
* more easy to extend to solve broader class of problem

Advantages of specialization

* ad hoc solutions can be found
* speed can be optimized for specific problem solution

Definition (simplified): Semantic Network

Method of knowledge representation using a graph whose nodes are objects and arcs are relationships.

Definition (general AI concept)

Representation in which:

* Lexically - there are nodes, arcs, and "application-specific" arc labels
* Semantically - the nodes and arcs denote "application-specific" entities

[

- Be able to describe a typical example.

- Be able to describe a specialization of this concept

(the ideas of inheritance in Java, for example).

]

Frames

Definition: Frame - data structure for representing stereotypical knowledge of some concept or object.  
Attributes of the frame concept.

* Frames have proven to be more useful in terms of OOD than other representations.
* The concept may be more restrictive than semantic nets; restrictions often make realizations (implementations) more practical.
* Concept was originally used to capture knowledge for linguistic semantic analysis.
* A frame can be viewed as a structuring of the Object-Attribute-Value triple.

General frame structure:

Frame name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Class name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Properties:

Property Value (Property aka attribute)

\_\_\_\_\_\_\_\_ \_\_\_\_\_

\_\_\_\_\_\_\_\_ \_\_\_\_\_

\_\_\_\_\_\_\_\_ \_\_\_\_\_

\_\_\_\_\_\_\_\_ \_\_\_\_\_

\_\_\_\_\_\_\_\_ \_\_\_\_\_

Properties are often called ``slots''.

Example: Bird Class frame structure:

Frame name \_\_Bird\_\_\_\_\_\_\_\_\_

Super-Class name \_\_Animal\_\_\_\_\_\_\_ (or vertebrate, etc.)

Properties:

Property Value (Property aka attribute)

\_Color\_\_ \_Unknown

\_No-wings \_2\_\_\_

\_Flies\_\_ \_True

\_Hungry\_ \_Unknown

\_Activity \_Unknown

Tweety Instance frame structure:

Frame (instance) name \_\_Tweety\_\_\_\_\_\_\_

Class name \_\_Bird\_\_\_\_\_\_\_\_\_

Properties:

Property Value (Property aka attribute)

\_Color\_\_ \_Unknown

\_No-wings \_2\_\_\_

\_Flies\_\_ \_True

\_Hungry\_ \_Unknown

\_Activity \_Unknown

Concepts related to frames (and OOD).

Definition: Inheritance - methodologies for creating new classes (or objects) based on existing classes (or objects).

Related definition: hierarchy -

Related definition: single inheritance - allowing for only one ``parent'' class in the inheritance hierarchy (*aka* derivation hierarchy.)

Related definition: multiple inheritance - the possibility of having multiple ``parent'' classes.

Definition: Slot - a component of a frame (like a field of a record.)

* facet - aspect of a slot (e.g., value, data type, dynamic vs. static, observer properties)
* daemon - a computed facet of a slot;

- if-needed

- if-changed

- if-added

- if-removed

Logic as knowledge representation in expert system

The formal logic systems use~ 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 disadvant~ge 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 bedescribed 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.

# CHAPTER 4

# Production system

A **production system** (or **production rule system**) is a computer program typically used to provide some form of [artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence), which consists primarily of a set of rules about behavior but it also includes the mechanism necessary to follow those rules as the system responds to states of the worl. Those rules, termed **productions**, are a basic [representation](https://en.wikipedia.org/wiki/Knowledge_representation) found useful in [automated planning](https://en.wikipedia.org/wiki/Automated_planning_and_scheduling), [expert systems](https://en.wikipedia.org/wiki/Expert_systems) and [action selection](https://en.wikipedia.org/wiki/Action_selection).

Productions consist of two parts: a sensory precondition (or "IF" statement) and an action (or "THEN"). If a production's precondition matches the current [state](https://en.wikipedia.org/wiki/State_(computer_science)) of the world, then the production is said to be *triggered*. If a production's action is [executed](https://en.wikipedia.org/wiki/Execution_(computers)), it is said to have *fired*. A production system also contains a database, sometimes called [working memory](https://en.wikipedia.org/wiki/Working_memory), which maintains data about current state or knowledge, and a rule interpreter. The rule interpreter must provide a mechanism for prioritizing productions when more than one is triggered

## Basic operation

Rule interpreters generally execute a [forward chaining](https://en.wikipedia.org/wiki/Forward_chaining) algorithm for selecting productions to execute to meet current goals, which can include updating the system's data or [beliefs](https://en.wikipedia.org/wiki/BDI_software_agent). The condition portion of each rule (*left-hand side* or LHS) is tested against the current state of the working memory.

In idealized or data-oriented production systems, there is an assumption that any triggered conditions should be executed: the consequent actions (*right-hand side* or RHS) will update the agent's knowledge, removing or adding data to the working memory. The system stops processing either when the user interrupts the forward chaining loop; when a given number of cycles has been performed; when a "halt" RHS is executed, or when no rules have LHSs that are true.

Real-time and expert systems, in contrast, often have to choose between mutually exclusive productions --- since actions take time, only one action can be taken, or (in the case of an expert system) recommended. In such systems, the rule interpreter, or [inference engine](https://en.wikipedia.org/wiki/Inference_engine), cycles through two steps: matching production rules against the database, followed by selecting which of the matched rules to apply and executing the selected actions.

## Matching production rules against working memory

Production systems may vary on the expressive power of conditions in production rules. Accordingly, the [pattern matching](https://en.wikipedia.org/wiki/Pattern_matching) algorithm which collects production rules with matched conditions may range from the naive—trying all rules in sequence, stopping at the first match—to the optimized, in which rules are "compiled" into a network of inter-related conditions.

The latter is illustrated by the [RETE](https://en.wikipedia.org/wiki/Rete_algorithm) algorithm, designed by [Charles L. Forgy](https://en.wikipedia.org/wiki/Charles_Forgy) in 1974, which is used in a series of production systems, called OPS and originally developed at [Carnegie Mellon University](https://en.wikipedia.org/wiki/Carnegie_Mellon_University) culminating in [OPS5](https://en.wikipedia.org/wiki/OPS5) in the early eighties. OPS5 may be viewed as a full-fledged programming language for production system programming.

## Choosing which rules to evaluate

Production systems may also differ in the final selection of production rules to execute, or *fire*. The collection of rules resulting from the previous matching algorithm is called the *conflict set*, and the selection process is also called a [*conflict resolution strategy*](https://en.wikipedia.org/wiki/Conflict_resolution_strategy).

Here again, such strategies may vary from the simple—use the order in which production rules were written; assign weights or priorities to production rules and sort the conflict set accordingly—to the complex—sort the conflict set according to the times at which production rules were previously fired; or according to the extent of the modifications induced by their RHSs. Whichever conflict resolution strategy is implemented, the method is indeed crucial to the efficiency and correctness of the production system. Some systems simply fire all matching productions.

**Rule Based System**

In [computer science](https://en.wikipedia.org/wiki/Computer_science), a **rule-based system** is used to store and manipulate knowledge to interpret information in a useful way. It is often used in [artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence) applications and research.

Normally, the term *rule-based system* is applied to systems involving human-crafted or curated rule sets. Rule-based systems constructed using automatic rule inference, such as [rule-based machine learning](https://en.wikipedia.org/wiki/Rule-based_machine_learning), are normally excluded from this system type.

**Applications**

A classic example of a rule-based system is the domain-specific [expert system](https://en.wikipedia.org/wiki/Expert_system) that uses rules to make deductions or choices. For example, an expert system might help a doctor choose the correct diagnosis based on a cluster of symptoms, or select tactical moves to play a game.

Rule-based systems can be used to perform [lexical analysis](https://en.wikipedia.org/wiki/Lexical_analysis) to [compile](https://en.wikipedia.org/wiki/Compiler) or interpret computer programs, or in [natural language processing](https://en.wikipedia.org/wiki/Natural_language_processing).

[Rule-based programming](https://en.wikipedia.org/wiki/Rule-based_programming) attempts to derive execution instructions from a starting set of data and rules. This is a more indirect method than that employed by an [imperative programming language](https://en.wikipedia.org/wiki/Imperative_programming_language), which lists execution steps sequentially

A typical rule-based system has four basic components:

* A list of rules or **rule base**, which is a specific type of [knowledge base](https://en.wikipedia.org/wiki/Knowledge_base).
* An [inference engine](https://en.wikipedia.org/wiki/Inference_engine) or [semantic reasoner](https://en.wikipedia.org/wiki/Semantic_reasoner), which infers information or takes action based on the interaction of input and the rule base. The interpreter executes a [production system](https://en.wikipedia.org/wiki/Production_system_(computer_science)) program by performing the following match-resolve-act cycle:[[2]](https://en.wikipedia.org/wiki/Rule-based_system#cite_note-2)
* Match: In this first phase, the left-hand sides of all productions are matched against the contents of working memory. As a result a conflict set is obtained, which consists of instantiations of all satisfied productions. An instantiation of a production is an ordered list of working memory elements that satisfies the left-hand side of the production.
* Conflict-Resolution: In this second phase, one of the production instantiations in the conflict set is chosen for execution. If no productions are satisfied, the interpreter halts.
* Act: In this third phase, the actions of the production selected in the conflict-resolution phase are executed. These actions may change the contents of working memory. At the end of this phase, execution returns to the first phase.
* Temporary [working memory](https://en.wikipedia.org/wiki/Working_memory).
* A [user interface](https://en.wikipedia.org/wiki/User_interface) or other connection to the outside world through which input and output signals are received and sent.

# Case-based reasoning

**Case-based reasoning** (**CBR**), broadly construed, is the process of solving new problems based on the solutions of similar past problems. An auto [mechanic](https://en.wikipedia.org/wiki/Mechanic) who fixes an [engine](https://en.wikipedia.org/wiki/Engine) by recalling another [car](https://en.wikipedia.org/wiki/Automobile) that exhibited similar symptoms is using case-based reasoning. A [lawyer](https://en.wikipedia.org/wiki/Lawyer) who advocates a particular outcome in a [trial](https://en.wikipedia.org/wiki/Trial_(law)) based on [legal](https://en.wikipedia.org/wiki/Legal) [precedents](https://en.wikipedia.org/wiki/Precedent) or a judge who creates [case law](https://en.wikipedia.org/wiki/Case_law) is using case-based reasoning. So, too, an [engineer](https://en.wikipedia.org/wiki/Engineer) copying working elements of nature (practicing [biomimicry](https://en.wikipedia.org/wiki/Biomimicry" \o "Biomimicry)), is treating nature as a database of solutions to problems. Case-based reasoning is a prominent type of [analogy](https://en.wikipedia.org/wiki/Analogy) solution making.

It has been argued that case-based reasoning is not only a powerful method for [computer reasoning](https://en.wikipedia.org/wiki/Computer_reasoning), but also a pervasive behavior in everyday human [problem solving](https://en.wikipedia.org/wiki/Problem_solving); or, more radically, that all reasoning is based on past cases personally experienced. This view is related to [prototype theory](https://en.wikipedia.org/wiki/Prototype_theory), which is most deeply explored in [cognitive science](https://en.wikipedia.org/wiki/Cognitive_science).

**Process**

Case-based reasoning has been formalized for purposes of [computer reasoning](https://en.wikipedia.org/wiki/Computer_reasoning) as a four-step process:

1. **Retrieve:** Given a target problem, retrieve from memory cases relevant to solving it. A case consists of a problem, its solution, and, typically, annotations about how the solution was derived. For example, suppose Fred wants to prepare blueberry [pancakes](https://en.wikipedia.org/wiki/Pancakes). Being a novice cook, the most relevant experience he can recall is one in which he successfully made plain pancakes. The procedure he followed for making the plain pancakes, together with justifications for decisions made along the way, constitutes Fred's retrieved case.
2. **Reuse:** Map the solution from the previous case to the target problem. This may involve adapting the solution as needed to fit the new situation. In the pancake example, Fred must adapt his retrieved solution to include the addition of blueberries.
3. **Revise:** Having mapped the previous solution to the target situation, test the new solution in the real world (or a simulation) and, if necessary, revise. Suppose Fred adapted his pancake solution by adding blueberries to the batter. After mixing, he discovers that the batter has turned blue – an undesired effect. This suggests the following revision: delay the addition of blueberries until after the batter has been ladled into the pan.
4. **Retain:** After the solution has been successfully adapted to the target problem, store the resulting experience as a new case in memory. Fred, accordingly, records his new-found procedure for making blueberry pancakes, thereby enriching his set of stored experiences, and better preparing him for future pancake-making demands.

# Inference engine

In the field of [artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence), **inference engine** is a component of the system that applies logical rules to the knowledge base to deduce new information. The first inference engines were components of [expert systems](https://en.wikipedia.org/wiki/Expert_system). The typical expert system consisted of a [knowledge base](https://en.wikipedia.org/wiki/Knowledge_base) and an inference engine. The knowledge base stored facts about the world. The inference engine applies logical rules to the knowledge base and deduced new knowledge. This process would iterate as each new fact in the knowledge base could trigger additional rules in the inference engine. Inference engines work primarily in one of two modes either special rule or facts: [forward chaining](https://en.wikipedia.org/wiki/Forward_chaining) and [backward chaining](https://en.wikipedia.org/wiki/Backward_chaining). Forward chaining starts with the known facts and asserts new facts. Backward chaining starts with goals, and works backward to determine what facts must be asserted so that the goals can be achieved.[[1]](https://en.wikipedia.org/wiki/Inference_engine#cite_note-Hayes-Roth_1983-1)

## Architecture

The logic that an inference engine uses is typically represented as IF-THEN rules. The general format of such rules is IF <logical expression> THEN <logical expression>. Prior to the development of expert systems and inference engines artificial intelligence researchers focused on more powerful [theorem prover](https://en.wikipedia.org/wiki/Theorem-prover) environments that offered much fuller implementations of [first-order logic](https://en.wikipedia.org/wiki/First-order_logic). For example, general statements that included [universal quantification](https://en.wikipedia.org/wiki/Universal_quantification) (for all X some statement is true) and [existential quantification](https://en.wikipedia.org/wiki/Existential_quantification) (there exists some X such that some statement is true). What researchers discovered is that the power of these theorem proving environments was also their drawback. It was far too easy to create logical expressions that could take an indeterminate or even infinite time to terminate. For example, it is common in universal quantification to make statements over an infinite set such as the set of all natural numbers. Such statements are perfectly reasonable and even required in mathematical proofs but when included in an automated theorem prover executing on a computer may cause the computer to fall into an infinite loop. Focusing on IF-THEN statements (what logicians call [Modus Ponens](https://en.wikipedia.org/wiki/Modus_ponens)) still gave developers a very powerful general mechanism to represent logic but one that could be used efficiently with computational resources. What is more there is some psychological research that indicates humans also tend to favor IF-THEN representations when storing complex knowledge.[[2]](https://en.wikipedia.org/wiki/Inference_engine#cite_note-2)

A simple example of Modus Ponens often used in introductory logic books is "If you are human then you are mortal". This can be represented in [pseudocode](https://en.wikipedia.org/wiki/Pseudocode" \o "Pseudocode) as:

Rule1: Human(x) => Mortal(x)

A trivial example of how this rule would be used in an inference engine is as follows. In *forward chaining*, the inference engine would find any facts in the knowledge base that matched Human(x) and for each fact it found would add the new information Mortal(x) to the knowledge base. So if it found an object called Socrates that was Human it would deduce that Socrates was Mortal. In *backward chaining*, the system would be given a goal, e.g. answer the question is Socrates Mortal? It would search through the knowledge base and determine if Socrates was Human and if so would assert he is also Mortal. However, in backward chaining a common technique was to integrate the inference engine with a user interface. In that way rather than simply being automated the system could now be interactive. In this trivial example if the system was given the goal to answer the question if Socrates was Mortal and it didn't yet know if he was human it would generate a window to ask the user the question "Is Socrates Human?" and would then use that information accordingly.

This innovation of integrating the inference engine with a user interface led to the second early advancement of expert systems: explanation capabilities. The explicit representation of knowledge as rules rather than code made it possible to generate explanations to users. Both explanations in real time and after the fact. So if the system asked the user "Is Socrates Human?" the user may wonder why she was being asked that question and the system would use the chain of rules to explain why it was currently trying to ascertain that bit of knowledge: i.e., it needs to determine if Socrates is Mortal and to do that needs to determine if he is Human. At first these explanations were not much different than the standard debugging information that developers deal with when debugging any system. However, an active area of research was utilizing natural language technology to ask, understand, and generate questions and explanations using natural languages rather than computer formalisms.[[3]](https://en.wikipedia.org/wiki/Inference_engine#cite_note-3)

An inference engine cycles through three sequential steps: *match rules*, *select rules*, and *execute rules*. The execution of the rules will often result in new facts or goals being added to the knowledge base which will trigger the cycle to repeat. This cycle continues until no new rules can be matched.

In the first step, match rules, the inference engine finds all of the rules that are triggered by the current contents of the knowledge base. In forward chaining, the engine looks for rules where the antecedent (left hand side) matches some fact in the knowledge base. In backward chaining, the engine looks for antecedents that can satisfy one of the current goals.

In the second step select rules, the inference engine prioritizes the various rules that were matched to determine the order to execute them. In the final step, execute rules, the engine executes each matched rule in the order determined in step two and then iterates back to step one again. The cycle continues until no new rules are matched.[[4]](https://en.wikipedia.org/wiki/Inference_engine#cite_note-4)

**CHAPTER 5**

**What is Knowledge Engineering?**

Imagine an education company wanting to automate the teaching of children in subjects from biology to computer science (requiring to capture the knowledge of teachers and subject matter experts) or Oncologists choosing the best treatment for their patients (requiring expertise and knowledge from information contained in medical journals, textbooks, and drug databases).

**Knowledge Engineering** is the process of imitating how a human expert in a specific domain would act and take decisions. It looks at the **metadata** (information about a data object that describes characteristics such as content, quality, and format), structure and processes that are the basis of how a decision is made or conclusion reached. Knowledge engineering attempts to take on challenges and solve problems that would usually require a high level of human expertise to solve. Figure 1 illustrates the knowledge engineering pipeline.

|  |
| --- |
| Figure 1: Knowledge engineering pipeline |
| ***Figure 1: Knowledge engineering pipeline*** |

**Knowledge Engineering Processes**

In terms of its role in **artificial intelligence (AI)**, knowledge engineering is the process of understanding and then representing human knowledge in **data structures**, **semantic models** (conceptual diagram of the data as it relates to the real world) and **heuristics** (rules that lead to solution to every problem taken in AI). **Expert systems**, and **algorithms** are examples that form the basis of the representation and application of this knowledge.

The knowledge engineering process includes:

* Knowledge acquisition
* Knowledge representation
* Knowledge validation
* Inferencing
* Explanation and justification

The interaction between these stages and sources of knowledge is shown in Figure 2.

|  |
| --- |
| Figure 2: Knowledge engineering processes |
| ***Figure 2: Knowledge engineering processes*** |

The amount of collateral knowledge can be very large depending on the task. A number of advances in technology and technology standards have assisted in integrating data and making it accessible. These include the **semantic web** (an extension of the current web in which information is given a well-defined meaning), **cloud computing** (enables access to large amounts of computational resources), and **open datasets** (freely available datasets for anyone to use and republish). These advances are crucial to knowledge engineering as they expedite data integration and evaluation.

**The Building Blocks of AI**

Artificial intelligence is not new and has been a computer science discipline since the 1950s. However, advances in technology and the reduction in costs of technology components have seen an acceleration towards true cognitive AI. AI is allowing people and companies to transform their businesses and invent new business models by adding intelligence to applications, processes, and equipment.

The key building blocks of AI are represented in Figure 3.

|  |
| --- |
| Figure 3: AI building blocks |
| ***Figure 3: AI building blocks*** |

### Knowledge Acquisition

Knowledge acquisition refers to the process of extracting, structuring, and organizing domain knowledge from domain experts into a program. A **knowledge engineer** is an expert in AI language and knowledge representation who investigates a particular problem domain, determines important concepts, and creates correct and efficient representations of the objects and relations in the domain.

Capturing domain knowledge of a problem domain is the first step in building an expert system. In general, the knowledge acquisition process through a knowledge engineer can be divided into four phases:

1.**Planning:** The goal is to understand the problem domain, identify domain experts, analyze various knowledge acquisition techniques, and design proper procedures.

2.**Knowledge extraction:** The goal is to extract knowledge from experts by applying various knowledge acquisition techniques.

3.**Knowledge analysis:** The outputs from the knowledge extraction phase, such as concepts and heuristics, are analyzed and represented in formal forms, including heuristic rules, frames, objects and relations, [semantic networks](https://www.sciencedirect.com/topics/computer-science/semantic-network), [classification schemes](https://www.sciencedirect.com/topics/computer-science/classification-scheme), [neural networks](https://www.sciencedirect.com/topics/computer-science/neural-networks), and fuzzy logic sets. These representations are used in implementing a prototype expert system.

4.**Knowledge verification:** The prototype expert system containing the formal representation of the heuristics and concepts is verified by the experts. If the knowledge base is incomplete or insufficient to solve the problem, alternative knowledge acquisition techniques may be applied, and additional knowledge acquisition process may be conducted.

Many knowledge acquisition techniques and tools have been developed with various strengths and limitations. Commonly used techniques include interviewing, protocol analysis, repertory grid analysis, and observation.

Interviewing is a technique used for eliciting knowledge from domain experts and design requirements. The basic form involves free-form or unstructured question–answer sessions between the domain expert and the knowledge engineer. The major problem of this approach results from the inability of domain experts to explicitly describe their reasoning process and the biases involved in human reasoning. A more effective form of interviewing is called **structured interviewing,** which is goal-oriented and directed by a series of clearly stated goals. Here, experts either fill out a set of carefully designed questionnaire cards or answer questions carefully designed based on an established domain model of the problem-solving process. This technique reduces the interpretation problem inherent in the unstructured interviewing as well as the distortion caused by domain expert subjectivity.

As an example, let's look at the interviewing process used in constructing GTE's COMPASS system (Prerau, 1990). COMPASS is an expert system that examines error messages derived from a telephone switch's self-test routines and suggests running of additional tests or replacing a particular component. The interviewing process in building COMPASS has an elicit–document–test cycle as follows:

1. Elicit knowledge from an expert.

2.Document the elicited knowledge in rules and procedures.

3.Test the new knowledge using a set of data:

(a)Have the expert analyze a new set of data.

(b)Analyze the same set of data using the documented knowledge.

(c)Compare the two results.

(d)If the results differ, find the rules or procedures that lead to the discrepancy and return to step 1 to elicit more knowledge to resolve the problem.

Protocol analysis is another technique of data analysis originated in clinical psychology. In this approach, an expert is asked to talk about his or her thinking process while solving a given problem. The difference from interviewing is that experts find it much easier to talk about specific problem instances than to talk in abstract terms. The problem-solving process being described is then analyzed to produce a structured model of the expert's knowledge, including objects of significance, important attributes of the objects, relationships among the objects, and inferences drawn from the relationships. The advantage of protocol analysis is the accurate description of the specific actions and rationales as the expert solves the problem.

Repertory grid analysis investigates the expert's mental model of the problem domain. First, the expert is asked to identify the objects in the problem domain and the traits that differentiate them. Then, a rating grid is formed by rating the objects according to the traits.

Observation involves observing how an expert solves a problem. It enables the expert to continuously work on a problem without being interrupted while the knowledge is obtained. A major limitation of this technique is that the underlying reasoning process of an expert may not be revealed in his or her actions.

Knowledge acquisition is a difficult and time-consuming task that often becomes the bottleneck in expert system development (Hayes-Roth et al., 1983). Various techniques have been developed to automate the process by using domaintailored environments containing well-defined domain knowledge and specific problem-solving methods (Rothenfluh et al., 1996). For example, OPAL is a program that expedites knowledge elicitation for the expert system ONCOCIN (Shortliffe et al., 1981) that constructs treatment plans for cancer patients. OPAL uses a model of the cancer domain to acquire knowledge directly from an expert. OPAL's domain model has four main aspects: **entities and relationships, domain actions, domain predicate,** and **procedural knowledge.** Based on its domain knowledge, OPAL can acquire more knowledge from a human expert and translate it into [executable code](https://www.sciencedirect.com/topics/computer-science/executable-code), such as production rules and finite state tables. Following OPAL, more general-purpose systems called PROTEGE and PROTEGE-II were developed (Musen, 1989). PROTEGE-II contains tools for creating [domain ontology](https://www.sciencedirect.com/topics/computer-science/domain-ontology) and generating OPAL-like knowledge acquisition programs for particular applications PROTEGE-II is a general tool developed by abstraction from a successful application, similar to the process from MYCIN to EMYCIN.

Another example of automated knowledge acquisition is the SALT system (Marcus and McDermott, 1989) associated with an expert system called Vertical Transportation (VT) for designing custom-design elevator systems. SALT assumes a propose-and-revise strategy in the knowledge acquisition process. Domain knowledge is seen as performing one of three roles: (1) proposing an extension to the current design, (2) identifying constraints upon design extension, and (3) repairing constraint violation. SALT automatically acquires these kinds of knowledge by interacting with an expert and then compiles the knowledge into production rules to generate a domain-specific knowledge base. SALT retains the original knowledge in a declarative form as a dependency network, which can be updated and recompiled as necessary.

To build a knowledge base, the knowledge can be either captured through knowledge engineers or be generated automatically by [machine learning techniques](https://www.sciencedirect.com/topics/computer-science/machine-learning-technique). For example, rules in rule-based expert systems may be obtained through a knowledge acquisition process involving domain experts and knowledge engineers or may be generated automatically from examples using [decision-tree](https://www.sciencedirect.com/topics/computer-science/decision-trees) [learning algorithms](https://www.sciencedirect.com/topics/computer-science/learning-algorithm). Casebased reasoning is another example of automated knowledge extraction in which the expert system searches its collection of past cases, finds the ones that are similar to the new problem, and applies the corresponding solutions to the new one. The whole process is fully automatic. An expert system of this type can be built quickly and maintained easily by adding and deleting cases. Automatic knowledge generation is especially good when a large set of examples exist or when no domain expert exists.

In addition to generating knowledge automatically, [machine learning methods](https://www.sciencedirect.com/topics/engineering/machine-learning-method) have also been used to improve the performance of the [inference engines](https://www.sciencedirect.com/topics/computer-science/inference-engines) by learning the importance of individual rules and better control in reasoning.

### Knowledge Acquisition Technique

At the heart of the process is the interview. The heuristic model of the domain is usually extracted through a series of intense, systematic interviews, usually extending over a period of many months. Note that this assumes the expert and the knowledge engineer are not the same person. It is generally best that the expert and the knowledge engineer not be the same person since the deeper the experts' knowledge, the less able they are in describing their logic. Furthermore, in their efforts to describe their procedures, experts tend to rationalize their knowledge and this can be misleading.

General suggestions about the knowledge acquisition process are summarized in rough chronological order below:

1. Observe the person solving real problems.
2. Through discussions, identify the kinds of data, knowledge and procedures required to solve different types of problems.
3. Build scenarios with the expert that can be associated with different problem types.
4. Have the expert solve a series of problems verbally and ask the rationale behind each step.
5. Develop rules based on the interviews and solve the problems with them.
6. Have the expert review the rules and the general problem solving procedure.
7. Compare the responses of outside experts to a set of scenarios obtained from the project's expert and the ES.

Note that most of these procedures require a close working relationship between the knowledge engineer and the expert.

## Practical Considerations

The preceding section provided an idealized version of how ES projects might be conducted. In most instances, the above suggestions are considered and modified to suit the particular project. The remainder of this section will describe a range of knowledge acquisition techniques that have been successfully used in the development of ES.

### Operational Goals

After an evaluation of the problem domain shows that an ES solution is appropriate and feasible, then realistic goals for the project can be formulated. An ES's operational goals should define exactly what level of expertise its final product should be able to deliver, who the expected user is and how the product is to be delivered. If participants do not have a shared concept of the project's operational goals, knowledge acquisition is hampered.

### Pre-training

Pre-training the knowledge engineer about the domain can be important. In the past, knowledge engineers have often been unfamiliar with the domain. As a result, the development process was greatly hindered. If a knowledge engineer has limited knowledge of the problem domain, then pre-training in the domain is very important and can significantly boost the early development of the ES.

### Knowledge Document

Once development begins on the knowledge base, the process should be well documented. In addition to tutorial a document, a knowledge document that succinctly state the project's current knowledge base should be kept. Conventions should be established for the document such as keeping the rules in quasi-English format, using standard domain jargon, giving descriptive names to the rules and including supplementary, explanatory clauses with each rule. The rules should be grouped into natural subdivisions and the entire document should be kept current.

### Scenarios

An early goal of knowledge acquisition should be the development of a series of well developed scenarios that fully describe the kinds of procedures that the expert goes through in arriving at different solutions. If reasonably complete case studies do not exist, then one goal of pre-training should be to become so familiar with the domain that the interviewer can compose realistic scenarios. Anecdotal stories that can be developed into scenarios are especially useful because they are often examples of unusual interactions at the edges of the domain. Familiarity with several realistic scenarios can be essential to understanding the expert in early interviews and the key to structuring later interviews. Finally, they are ultimately necessary for validation of the system.

### Interviews

Experts are usually busy people and interviews held in the expert's work environment are likely to be interrupted. To maximize access to the expert and minimize interruptions it can be helpful to hold meetings away from the expert's workplace. Another possibility is to hold meetings after work hours and on weekends. At least initially, audiotape recordings ought to be made of the interviews because often times notes taken during an interview can be incomplete or suggest inconsistencies that can be clarified by listening to the tape. The knowledge engineer should also be alert to fatigue and limit interviews accordingly.

In early interviews, the format should be unstructured in the sense that discussion can take its own course. The knowledge engineer should resist the temptation to impose personal biases on what the expert is saying. During early discussions, experts are often asked to describe the tasks encountered in the domain and to go through example tasks explaining each step. An alternative or supplemental approach is simply to observe the expert on the job solving problems without interruption or to have the expert talk aloud during performance of a task with or without interruption. These procedures are variations of protocol analysis and are useful only with experts that primarily use verbal thought processes to solve domain problems.

For shorter term projects, initial interviews can be formalized to simplify rapid prototyping. One such technique is a structured interview in which the expert is asked to list the variables considered when making a decision. Next the expert is asked to list possible outcomes (solutions) from decision making. Finally, the expert is asked to connect variables to one another, solutions to one another and variables to solutions through rules.

A second technique is called twenty questions. With this technique, the knowledge engineer develops several scenarios typical of the domain before the interview. At the beginning of the interview, the expert asks whatever questions are necessary to understand the scenario well enough to determine the solution. Once the expert begins the questions, the expert is asked to explain why each question is asked. When the interviewer perceives a rule, he interrupts and restates the rule to ensure that it is correct.

A third technique is card sorting. In this procedure, the knowledge engineer prepares a stack of cards with typical solutions to problems in the domain. The expert is asked to sort the cards according to some characteristic important to finding solutions to the problem. After each sort, the expert is asked to identify the sorting variable. After each sort, the expert is asked to repeat the process based on another variable. Note that this technique is usually not as effective as the 2 previous.

In large projects, later interviews cannot be expected to be as productive as early interviews. Typically, later interviews should become increasingly structured and follow a cyclical pattern where bits of knowledge are elicited, documented and tested. During this phase of knowledge acquisition, the interviewer must begin methodically to uncover the more subtle aspects of the knowledge. Typically, this process is based on scenarios. By modifying the scenarios in different ways, the interviewer can probe the expert's sensitivity.

During interviews, it may be helpful to work at a whiteboard to flexibly record and order the exact phraseology of rules or other representations. It may also be helpful to establish recording conventions for use such as color coding different aspects of a rule and using flags to note and defer consideration of significant but peripheral details.

Structured interviews should direct the course of a meeting to accomplish specific goals defined in advance. For instance, once a prototypic knowledge base is developed, the expert can be asked to evaluate it line by line. Other less obvious structures can be imposed on interviews, such as asking the expert to perform a task with limited information or during a limited period of time. Even these structured interviews can deviate from the session's intended goals. Sometimes such deviations show subtleties in the expert's procedures and at other times the interview simply becomes sidetracked, requiring the knowledge engineer to redirect the session.

## Questionnaires

When specific information is needed, a questionnaire can sometimes be used effectively. Questionnaires are generally used in combination with other techniques such as interviews.

## Decision Trees

Decision trees are widely recognized to be useful tools for the knowledge engineer in prototyping knowledge representations. In addition, they can be useful in knowledge acquisition on several different levels. Some knowledge engineers have found that experts can more readily relate to decision trees than rules.

## Rule Development

Although complex representation techniques might eventually be used, rules are generally easier to use for characterizing knowledge during knowledge acquisition. Prototypic rules should be developed as soon as possible to serve as a focal point for directing the course of the knowledge acquisition process. The initial knowledge base can be developed from written materials or from example cases described by the expert during early unstructured interviews. Initial rules should be treated as approximations and their wording should be general to avoid pressuring the expert. As additional cases are described during interviews, the rule base can be expanded. Once a stable rule base begins to develop, it can provide feedback for structuring interviews. Initially the rules and procedures can be traced through by hand with the expert considering each step. The same pattern of tracing through rules should continue once a version of the knowledge base is developed on a computer and it frequent use should become part of the process.

CHAPTER 6

BUILDING EXPERT SYSTEM

**LIFE CYCLE DEVELOPMENT STEPS**

Existing life-cycles for the development of traditional information systems are shown to be inadequate for addressing expert system requirements. A life cycle for expert systems is constructed, outlining the tasks and activities to be performed at each stage of system development. The life cycle highlights the role of alternative development paradigms and the importance of social and organizational characteristics in system transfer to users. The operationalization of the life-cycle is illustrated through a case study, describing the development process and major architectural components of an expert system that configures air-conditioning units. Several important issues encountered in transferring the system from test to production mode and in system evaluation are highlighted. Such a life-cycle approach can enhance project management for expert systems.

EXPERT SYSTEM LANGUAGE

Expert systems have been constructed using various general-purpose programming languages as well as specific tools. LISP and PROLOG have been used widely. OPS-5 has also been popular among rule-based programmers. OPS is a product of the Instructable Production System Project at CMU in 1975. It was used to create one of the classic expert systems called R1 (later called XCON) to help DEC configure VAX computers. C and C++ have been used in applications when execution speed is of importance. C Language Integrated Production System (CLIPS) is a popular development environment for building rule- and object-based expert systems. CLIPS is written in C for portability and speed. JAVA is relatively new and is gaining popularity in building expert systems mostly due to the desirable attribute of portability on different computer platforms.

Compared to general-purpose programming languages, shells provide better tools for fast design and implementation of expert systems. An expert system **shell** is a programming environment that contains the necessary utilities for both developing and running an expert system. For example, since an expert system shell provides a build-in [inference engine](https://www.sciencedirect.com/topics/computer-science/inference-engines), the developer can focus on inputting problem-specific knowledge into the knowledge base. By using expert system shells, the programming skills required to build an expert system become minimal, and the design and implementation time can be greatly reduced.

EMYCIN (Buchanan and Shortliffe, 1984) is an example of shells. The tools it provides include (a) an abbreviated rule language that is easier to read than LISP and more concise than the English subset used by MYCIN; (b) an indexing and grouping scheme for rules; (c) a goal-driven inference engine; (d) a [communication interface](https://www.sciencedirect.com/topics/engineering/communication-interface) between the program and the end user; (e) an interface between the system designer and the program, which supports displaying and editing rules, editing knowledge in tables, and running selected rules sets on a problem; and (f) facilities for monitoring the behavior of the program, such as explaining how a conclusion is reached, comparing the results of the current run with correct results and exploring the discrepancies, and reviewing conclusions about a stored library of cases.

General-purpose shells are offered to address a broad range of problems. Many domain-specific tools have also been developed to provide special features to support the development of expert systems for specific domains. The introduction and widespread use of expert system shells promotes the successful applications of expert systems to a wide range of tasks.

Most of the expert systems now completed were developed using LISP; indeed, the evolution of LISP itself has facilitated the development of expert systems. Moreover, together with LISP features, such as the pretty-printing of programs and functions for correcting spelling mistakes, expert systems have been constructed in which the interface with the human user has been well .There exist, of course, expert systems which have been developed in other languages, such as FORTRAN [2], but LISP remains the primary language for most work in both expert systems and artificial intelligence.

**EXPERT SYSTEM MODEL**

An expert system that generates digital system design from high-level specifications is described. It is called MBESDSD, short for model-based expert system for automated digital systems. The system is based on a three-phase model of digital system design. First, the high-level behavioral specifications are translated into a sequence of primitive behavioral operations. Next, these primitive operations are grouped to form intermediate-level behavioral functions. Finally, structural function modules are selected to implement these functions. The advantages of this model-based reasoning technique is that more design solutions are possible and the design is easier to optimize. The design of a popular peripheral device, the ART asynchronous receiver/transmitter, is used to demonstrate the proposed design process using MBESDSD.

**KNOWLEDGE REPRESENTATION SYSTEMS**

Knowledge representation systems, also called expert systems, are computerized models that capture the knowledge of one or more human experts and store it in the framework that is most appropriately suited to the reasoning processes that the experts use in their problem-solving behavior. Such systems are created by a specialized systems analyst called a knowledge engineer, whose task is to interview the expert and/or observe his problem-solving behavior, then determine the most appropriate form(s) of knowledge representation to model the expert's problem-solving techniques. This process, called knowledge acquisition, is perhaps the most difficult and time-consuming aspect of expert systems development.

It requires both technical and people skills on the part of the knowledge engineer, who must establish rapport with the domain expert, maintain a productive relationship during the interviewing process, and recognize the required mapping from the expert's explanations to the appropriate knowledge representation. The knowledge engineer then encodes the expert's knowledge into a knowledge base, which is a repository of the expert's knowledge in a particular representational structure. Some of the most common knowledge representations are described below.

In addition to the knowledge base, an expert system includes an automated reasoning mechanism called an inference engine that performs calculations and/or logical processes to produce the results of a particular problem-solving session. The explanation facility of an expert system provides the user with an explanation of the reasoning process that was used to achieve the conclusion or recommendation. Each knowledge representation has a corresponding inference technique. Three very common knowledge representations are rule-based systems, frame-based systems, and case-based systems.

# CHAPTER 7

# CURRENT RESEARCH TRENDS

An expert system, also known as a knowledge based system, is a computer program that contains some of the subject-specific knowledge of one or more human experts. This class of program was first developed by researchers in artificial intelligence during the 1960s and 1970s and applied commercially throughout the 1980s. The most common form of expert systems is a program made up of a set of rules that analyze information (usually supplied by the user of the system) about a specific class of problems, as well as providing mathematical analysis of the problem(s), and, depending upon their design, recommend a course of user action in order to implement corrections. It is a system that utilizes what appear to be reasoning capabilities to reach conclusions. This new book presents important research on in this dynamic field.

* ... Moreover, ESs have been very successful in modelling real-world decision-making problems which conventional programming techniques are unable to handle (Giarratano and Riley 2005). Some of the types of tasks that have been addressed using ES include interpretation, prediction, diagnosis, design, planning, monitoring, control and repair across many different fields including medicine, engineering and financial analysis (Liao 2005;Tyler 2007). ...

... Despite these criticisms, ES applications still continue to appear in the literature, either as separate collections (e.g. Tyler 2007) or in relevant international journals such as ES with Applications, Knowledge-Based Systems and the International Journal of Knowledge-Based and Intelligent Engineering Systems. The wide-ranging applications that appear regularly in these journals clearly indicate that the field of ES is strong across many disciplines and continues to grow. ...

* ... Trends towards integration of research in different fields of artificial intelligence had most clearly manifested at the turn of the XX and the XXI centuries and made it necessary to combine semantically different objects, models, methodologies, concepts and technologies. As a result, new intelligent system architectures were emerged, in particular integrated expert systems (IES),
* ... AI techniques such as rule-based expert systems offer better ways of representing and sharing hard-won knowledge. Expert systems are computer programs that are used to solve procedural problems in the manner a human expert can (Tyler, 2007). Expert systems assist persons with less experience in performing their duties in a manner that is similar to the way a human expert can carry out the tasks. ...
* ... AI techniques such as rule-based expert systems offer better ways of representing and sharing hard-won knowledge. Expert systems are computer programs that are used to solve procedural problems in the manner a human expert can (Tyler, 2007). Expert systems assist persons with less experience in performing their duties in a manner that is similar to the way a human expert can carry out the tasks. ...

----END------