Griffith University

School of Computing and Information Technology

Domain: Expert Systems Design and Development

The Evolution of Expert Systems

available electronically at:

http://www.cit.gu.edu.au/~noran

Table of Contents.

1 Introduction.

1.1 Artificial Intelligence and Expert Systems.

Mankind has always been asking questions. It is a way to relate to the world and to evolve: '*I think, therefore I am*'. Some questions though are difficult to answer. Take for example: 'What is the nature of knowledge ?', 'Can knowledge be represented ?'. In trying to answer these questions, Artificial Intelligence (A.I.) is an important contributor.

1.1.1 Artificial Intelligence.

But, *what is A.I.* ? One of the many definitions may be:

'The branch of computer science concerned with the automation of intelligent behaviour'.

As a computer science, A.I. is based on computer science specific principles, such as data structures, algorithms, languages and programming techniques. In contrast with these well defined terms, questions such as 'what *is* intelligence ?' or 'may intelligence be achieved on a computer or just within a biological existence?' are yet unanswered.

An important fact though is that A.I., like any other science, is a human endeavour and therefore may also be defined as *'the collection of problems and methodologies studied by artificial intelligence researchers'* (Ref.1, p.2).

A.I. has many areas of interest: robotics, vision, natural language understanding and semantic modelling, speech, artificial neural systems and parallel processing, automated reasoning and theorem proving, *expert systems,* game playing and human performance modelling. While each and every A.I. domain is both challenging and fascinating, we will have to limit the topic of this paper to *expert systems*.

One of the major roots of the expert systems is in the *cognitive science*: the way humans process information (i.e., the way they think and solve problems).

1.1.2 Expert System.

So, *what is an expert system ?*

An expert system is *a computer system that emulates the decision-making ability of a human expert, i.e. it acts in all respects as its human counterpart*.

The term *expert* may be misleading. In the early days expert systems only contained *expert knowledge*. Presently however, *any* system using expert system *technology* (even if not containing highly specialised expertise in a certain domain) is called an expert system. Therefore, the term *knowledge-based system* is more appropriate, although most people use the term *expert system* because it is shorter.

Expert systems have emerged from early work in problem solving, mainly because of the importance of domain-specific knowledge. A human expert's knowledge is specific to a problem *domain*. In much the same way, expert systems are designed to address a specific domain, called the *knowledge domain*.

Fig. 0. 1 shows the concept of a knowledge-based expert system. The expert system receives facts from the user and provides *expertise* in return. The main components of the expert system (invisible from the outside) are the *knowledge base* and the *inference engine*. The inference engine may *infer* conclusions (solutions) from the knowledge base, based on the 'facts' supplied by the user.

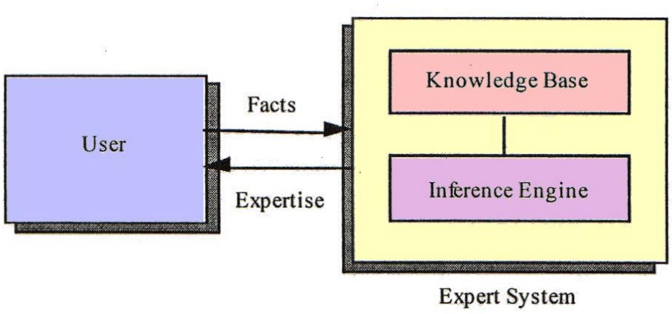


Fig. 0. 1 Basic concept of an expert system *(Ref.1, p.3)*

Expert systems are suitable for solving the *ill-structured* (or not entirely defined) type problems, which are usually difficult or impossible to solve by 'conventional' methods (e.g. procedural programming).

1.2 Motivation.

In the development of any tool, method, etc, a question will inevitably be asked: Why develop and use it ? What is it good for ?

In order to determine the advantages of a generic expert system, we will have to evaluate it with the only comparison term available: the human expert.

- availability: On any suitable computer hardware;
- cost: The per-user cost is greatly lowered;
- danger: May be used in environments hazardous for humans;
- permanence: will last indefinitely (depending on the hardware);
- multiplicity: more expert systems may co-operate to increase the level of expertise;
- reliability: provide a second opinion to human experts, mediate opinions;
- explanation: the expert system may always explain how it reached the conclusion;
- response: may provide fast or real-time response for critical applications;
- emotional: in real-time or emergency situations, the expert system will always provide an unemotional and complete response;
- database: may be used to intelligently access a database (data mining);
- tutoring: may be used as an intelligent tutor: explain reasoning, etc.

The most obvious shortcomings are that an expert system has to be developed, trained, updated and that it cannot capture nor explain 'deep' knowledge (understanding of underlying causes). Also, unless carefully designed, an expert system may entirely reflect a single human's expertise, which would defeat most of the advantages enumerated above. The limitations of the expert systems will be discussed later on in this paper.

1.3 The objective of this paper.

This paper is intended as a primer in the area of Artificial Intelligence and especially Expert Systems. We have started by giving a motivation, some definitions and basic concepts. In the following sections we will detail some expert systems features and trace their development from the early days through to present day and future trends.

⇧ Back to Table Of Contents.

2 Expert Systems Primer.

In order to understand the development of the expert systems, the reader needs to build a minimum of *domain knowledge* in the area of expert systems. Therefore we will start by presenting some basic and advanced concepts, followed by desired features and application domains of expert systems. Armed with these notions, we will then be ready to follow the evolution of the expert systems.

2.1 General concepts.

2.1.1 Rule-based expert systems.

Knowledge representation in expert systems may be *rule-based* or encapsulated in *objects*. The rule-based approach uses IF-THEN type *rules* and it is the method currently used in constructing expert systems. IF-THEN rules take the following form:

IF there is a flame THEN there is a fire.

The modern rule-based expert systems are based on the *Newel and Simon* model of human problem solving in terms of long-term memory (rules), short-term memory (working memory) and cognitive processor (inference engine).

Elaborate expert systems may be based on thousands of rules (e.g. XCON/R1 system from Digital Equipment Corporation, used for configuring computers) and surpass a human expert in a particular field. However, even smaller sized expert systems, based on several hundred rules may be extremely efficient in very specialised areas.

While a knowledge-based system may rely on knowledge commonly available, a true 'expert' system will be based on unwritten expertise, acquired from a human expert.

In the conditions where no algorithm is available to solve a particular problem, a *reasonable* solution is the best we can expect from an expert (system or human). The expert system will *infer* a solution from the facts provided by the user and the rules in the knowledge base. Therefore, it should be able to *explain* the reasoning employed to achieve the solution. The *explanation facility* is an important feature of the rule-based expert systems, since it provides a mechanism for a human to follow and check the correctness of the solution achieved by the expert system. A further enhancement to this facility is the availability of *what-if* scenarios (employing *hypothetical reasoning* questions), where the user may examine the outcomes of several possible situations.

2.1.2 Development of an expert system.

The process of building an expert system is commonly known as *knowledge engineering*. This implies knowledge acquisition from a human or other source and coding it into the knowledge base of the expert system (refer **Fig. 1. 1**).

The main phases in the knowledge engineering process are:

- The *dialog* process represented in **Fig. 1. 1** is similar to the task of a system designer discussing the requirements of the program with the client, in conventional programming. After acquiring knowledge from the human expert, the knowledge engineer has to *explicitly* code it into the expert system knowledge base.
- After the coding stage, the human expert evaluates the expert system and gives feedback/critique to the knowledge engineer.

- The knowledge engineer alters the knowledge base in order to reflect the human expert's comments.

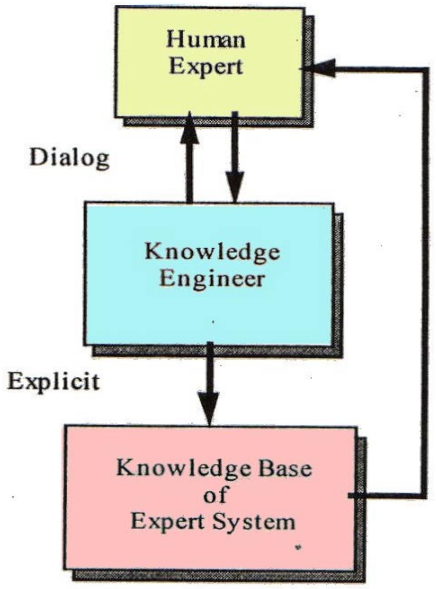


Fig. 1. 1 Basic concept of an expert system *(Ref.1, p.7)*

This process will iterate until the human expert finds the expert system satisfactory for the envisaged purpose.

Some rule-based systems may display the capability to *learn* rules by example (*rule induction*), creating rules from tables of data. Learning is a complex process, hindered by ambiguities, duplications and inconsistencies in the human expert knowledge.

2.1.3 Limitations of expert systems.

Although the expert system technology has progressed substantially, it would be a mistake to overestimate the abilities of the technology. Deficiencies of the current expert systems include:

- the lack of *causal* (or *deep*) knowledge. The expert systems do not have a real understanding of the causes and effects in a system, mainly because it is much easier to program expert systems with *shallow knowledge*, based on empirical and heuristic knowledge. Designing an expert system based on basic structures, functions and behaviours of objects takes a lot more effort and the resulting system is a lot more complex (and more difficult to maintain / upgrade)
- dealing with uncertainty. Human experts recognise the limits of their knowledge and can qualify their expertise when the problem reaches their so-called *limit of ignorance*. An expert system, unless this problem is *explicitly* catered for, will make confident recommendations even in the presence of incomplete and/or inaccurate data. Conclusion: the expert system's solution must *degrade gracefully*, just like in the case of a human expert.
- the current expert systems cannot make *analogies*, i.e. cannot generalise their knowledge in order to reason about new situations in the way that people can. Rule induction may help the system achieve *only some* types of new knowledge.

But there’s also good news. Expert systems, even limited in the present form, have still been able to solve problems that conventional programming methodologies were unable to scope with. By fully understanding the capabilities and limitations of the expert system technology, we can make appropriate use of it.

2.2 Advanced concepts.

2.2.1 Languages, tools and shells.

In defining a problem, a very important decision is the way to model it. If the problem may be modelled by conventional programming, that paradigm is usually preferred because of the vast amount of experience available in that domain. However some problems can only be solved via a *non-procedural* programming paradigm. In this case, *expert system languages* will have to be used.

Expert system languages usually address a smaller range of problems in a more efficient and natural way, however they are not generally suitable for general-purpose programming. This is particularly true for the specialised AI languages, such as IPL-II, SAIL, CONNIVER, Smalltalk, etc. The commonly used AI languages may have extensions that make them usable for a wide variety of problems. For example, one of the current Prolog versions, *Visual Prolog*, can be used for expert systems, but also for database and Web programming.

The main difference between procedural languages and expert system languages is the *focus of representation*. Procedural languages provide flexible and robust ways of representing *data*, (arrays, records, linked lists, etc) and even provide for data abstraction (modules, packages). However, the data and methods to manipulate it however are tightly interlaced. Expert systems languages by contrast allow *two levels* of abstraction: *data abstraction* and *knowledge abstraction*. For example, in the rule-based expert systems, the *facts* (data abstraction) are separated from the *rules* (knowledge abstraction). The data – knowledge separation allows for less control of the execution sequence (e.g. typically a separate piece of code, the *inference engine*, applies knowledge to the data).

In order to avoid confusion, the following terms have to be defined:

- language: translator of command written in a specific syntax. An expert system language provides an *inference engine* to execute statements of the language;
- tool: language *plus* associated utility programs for debugging, development, deploying applications. Examples: text editors, (cross) compilers, code generators (which can generate code from tables). Some tools may be integrated with all its utility programs under a single integrated user interface;
- shell: a special purpose tool comprising a complete expert system, however *without* the knowledge base. The user must only supply the knowledge base. Example: *EMYCIN*, obtained by removing the medical database from the *MYCIN* expert system.

2.2.2 Components of a rule-based expert system.

The structure of a rule-based expert system is shown in **Fig. 1. 2**. In this case, the domain knowledge needed to solve problems is contained in the knowledge base in the form of *rules*.

Components of the rule-based expert system:

- user interface: it is the user – expert system means of communication;
- explanation facility: explains the expert system reasoning to the user;
- working memory: a *global database* of facts used by the rules;
- inference engine: decides which rule is satisfied by the supplied facts, prioritise the rules and execute the highest priority rule;

- agenda: the prioritised list of rules whose patterns are satisfied by facts (or objects) in the working memory;
- (*optional*) knowledge acquisition facility: a way for the user to directly enter knowledge in the system, without the need for explicit knowledge coding.

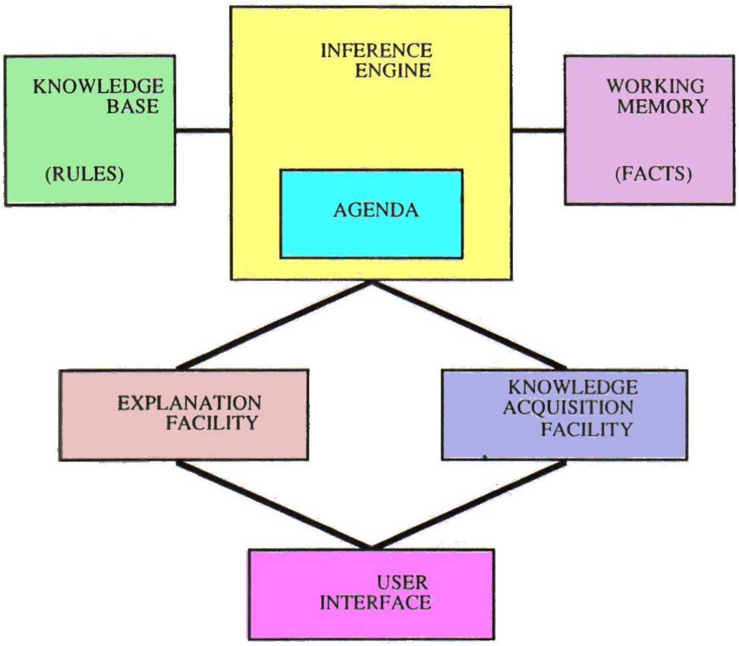


Fig. 1. 2 Structure of a Rule based expert system *(Ref.1, p.24)*

A rule whose patterns are satisfied by the facts are said to be *activated*, or *instantiated*. Given more rules are instantiated at the same time, the inference engine must select one rule for *firing* (termed by analogy with a nerve cell - *neuron*). Like a neuron, once the rule has fired, it may not fire again for a certain period of time. This behaviour is aimed at preventing trivial loops and is called *refraction*.

2.2.3 The production rules.

In rule-based expert systems, the knowledge base is also known as the *production memory*. The rules contained within the knowledge base are called *production rules* and seen as a way of transforming a string of symbols into another string of symbols (refer Chapter 2).

Rules may be expressed in an IF…THEN… pseudocode. The section between the IF and THEN parts is called the antecedent, conditional part, pattern part, of *LHS* (left hand side). The THEN part of the rule is a list of actions to be performed when the rule *fires* (i.e. *LHS* is satisfied and the rule has enough priority to be executed by the inference engine). This part of the rule is called the consequent or the *RHS* (right hand side).

When multiple rules are selected for firing, they are activated and put on the agenda in descending priorities order. However, some rules' firing may determine a *change* in system status, so that other rules waiting on the agenda no longer have their *LHS* (pattern) satisfied. In this case, they are simply removed from the agenda.

2.3 Features.

In order to perform within the intended scope, expert systems must contain a set of desired *features*. These features must ensure that the expert system is appropriately accepted and utilised by the intended end users. An expert system that is not applied and / or acknowledged by the users is a useless exercise even if it is a very good and comprehensive tool.

2.3.1 Essential features.

A set of *essential* features expected in an expert system would be:

- high performance. The quality of the advice given by the system must be very high – comparable or better than a human expert;
- adequate response time. Again, the expert system must respond in an amount of time comparable or better than a human expert. In case of real time processes, the *time constraint* becomes especially important;
- reliability. The expert system must be immune to crashes and stable, or its usability and acceptance will be greatly reduced;
- understandable. The system must be able to provide an explanation for the conclusions / solutions it reaches. This feature is extremely important for the following reasons:
    - ⇒ solution justification – needed due to the great potential for harm (human life and property may depend on the answers of an expert system);
    - ⇒ expert system development – debugging misunderstandings between the knowledge engineer and the expert;
    - ⇒ behaviour checking – debug any unforeseen interactions in the expert system, seen as a parallel program with the rules acting as independent knowledge processors;
- flexible. The system must be easy to maintain: add, modify, and delete change.

2.3.2 Advanced features.

The explanation facility within the expert system may be simple or sophisticated, according to the size / intended use of the system. The explanation facility may provide one or more of the following:

- list all the *pro* and *con* reasons for a particular hypothesis;
- list all hypothesis that may explain the provided fact;
- explain all consequences of a particular hypothesis;
- use a *what-if* approach (prognosis on what will occur if a hypothesis is true);
- justify the questions asked by the expert system – maybe provide a way for the user to guide the system towards achieving a solution more efficiently. The user input has to be very carefully balanced though;
- justify the knowledge of the program on user request; a particular case is quoting a *meta-rule* for the justification of a particular rule. Meta-rules are knowledge *about* rules.

2.4 Application domains.

Expert systems can only be effective in certain *domains*. Therefore selecting the appropriate paradigm for the problem is *essential*. In many cases conventional programming will solve the problem. The appropriate domain for an expert system however depends on a few critical *factors*:

- If the problem be solved reasonably through conventional programming, then an expert system is not the best choice. However for situations where there is no efficient algorithmic solution (*ill-structured*), expert systems are best suited.

When dealing with ill-structured problems, there is a possibility that the system may follow an algorithmic solution after all. This may happen if the knowledge engineer sets the priorities of too many rules, so the systems unknowingly mirrors an algorithmic solution. An expert system with strong control structure (displaying opportunistic behaviour - favouring certain paths) often indicates a concealed algorithm that could be represented through conventional programming.

- The expert system must be *clearly delimited* and is expected to 'know' its own capabilities. By adding domains to the knowledge base, the expert system becomes more and more knowledgeable, however its complexity increases too.

Coordinating multiple expert systems (in the HEARSAY II and III systems, *Ref 1 p.19*) has proven to be a complex task, presently suited more for research than for a deliverable (commercial) product.

- There must be a need and desire for an expert system. As previously mentioned, it is meaningless to build an expert system if there is no one willing to make use of it. The expert system needs to be accepted as a helping tool, or it will be rejected even if it is needed.

Expert systems are *new* technology, so gaining management support is crucial. This is mainly because of uncertainty of the achievements and the lack of widespread expert system knowledge.

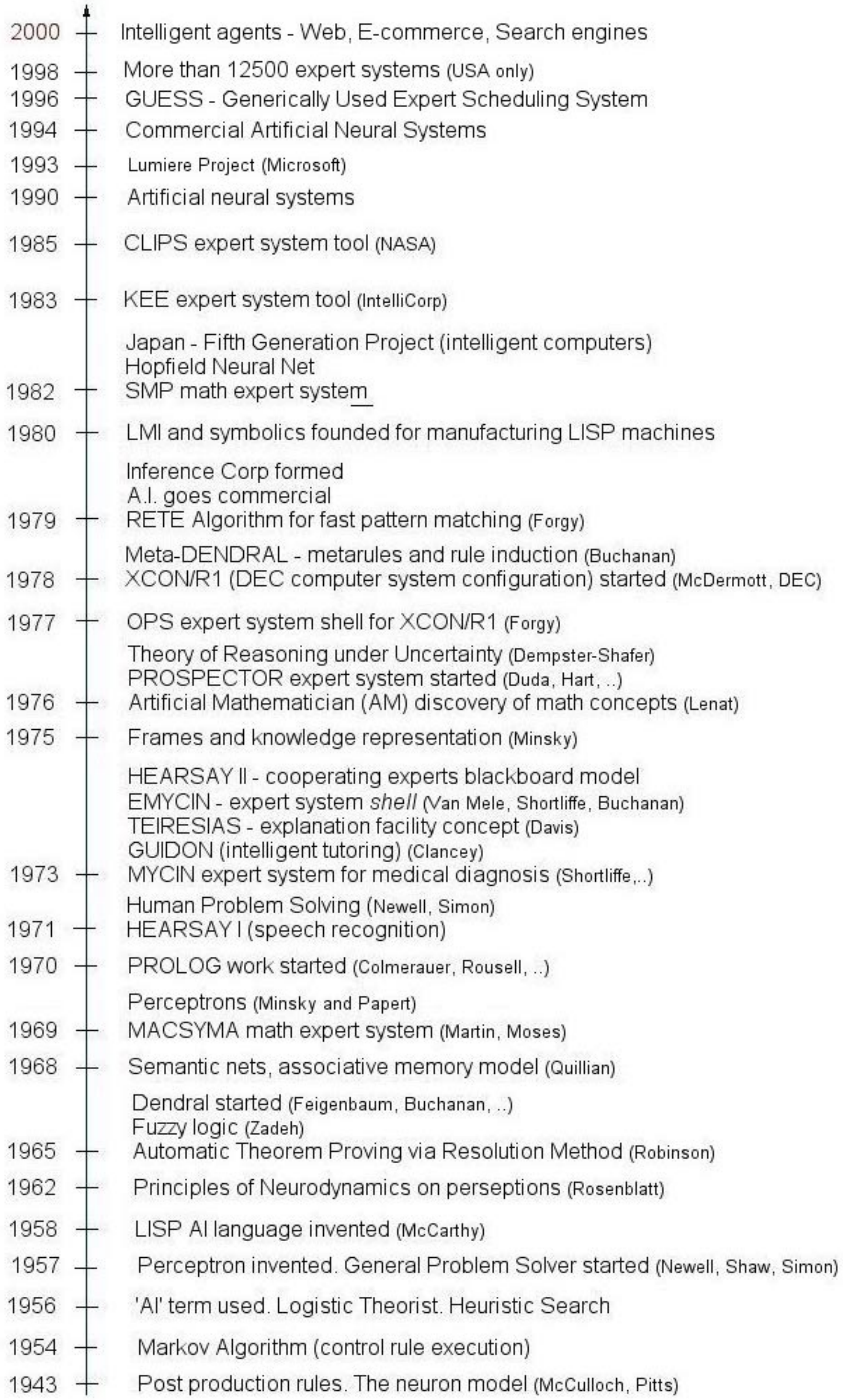
- There must be a human expert willing to co-operate. At least one, but not too many, since the system may become a repository of contradictory rules (as we know experts rarely agree in opinions);
- The human expert knowledge must be understood by the knowledge engineer. The knowledge engineer must understand what the human expert is talking about, because he is the middle tier between human expert and expert system. He has to explicitly *code* the knowledge into the system.
- Expert systems are especially suitable when the human expert's knowledge is mainly *heuristic* (from experience) and uncertain. The expert's knowledge may be trial-and-error approach rather than based on logic and algorithms (which may be better modelled by conventional programming).

⇧ Back to Table Of Contents.

3 Expert Systems Evolution.

3.1 Timeline.

The expert systems technology is based on a wide background. Fig. 2. 1 presents the major milestones in the development of the expert systems.



2000 Intelligent agents - Web, E-commerce, Search engines
1998 More than 12500 expert systems (USA only)
1996 GUESS - Generically Used Expert Scheduling System
1994 Commercial Artificial Neural Systems
1993 Lumiere Project (Microsoft)
1990 Artificial neural systems
1985 CLIPS expert system tool (NASA)
1983 KEE expert system tool (IntelliCorp)
Japan - Fifth Generation Project (intelligent computers)
Hopfield Neural Net
1982 SMP math expert system
1980 LMI and symbolics founded for manufacturing LISP machines
Inference Corp formed
A.I. goes commercial
1979 RETE Algorithm for fast pattern matching (Forgy)
Meta-DENDRAL - metarules and rule induction (Buchanan)
1978 XCON/R1 (DEC computer system configuration) started (McDermott, DEC)
1977 OPS expert system shell for XCON/R1 (Forgy)
Theory of Reasoning under Uncertainty (Dempster-Shafer)
PROSPECTOR expert system started (Duda, Hart, ..)
1976 Artificial Mathematician (AM) discovery of math concepts (Lenat)
1975 Frames and knowledge representation (Minsky)
HEARSAY II - cooperating experts blackboard model
EMYCIN - expert system *shell* (Van Mele, Shortliffe, Buchanan)
TEIRESIAS - explanation facility concept (Davis)
GUIDON (intelligent tutoring) (Clancey)
1973 MYCIN expert system for medical diagnosis (Shortliffe,..)
Human Problem Solving (Newell, Simon)
1971 HEARSAY I (speech recognition)
1970 PROLOG work started (Colmerauer, Rousell, ..)
Perceptrons (Minsky and Papert)
1969 MACSYMA math expert system (Martin, Moses)
1968 Semantic nets, associative memory model (Quillian)
Dendral started (Feigenbaum, Buchanan, ..)
Fuzzy logic (Zadeh)
1965 Automatic Theorem Proving via Resolution Method (Robinson)
1962 Principles of Neurodynamics on perseptions (Rosenblatt)
1958 LISP AI language invented (McCarthy)
1957 Perceptron invented. General Problem Solver started (Newell, Shaw, Simon)
1956 'AI' term used. Logistic Theorist. Heuristic Search
1954 Markov Algorithm (control rule execution)
1943 Post production rules. The neuron model (McCulloch, Pitts)

Fig. 2. 1 Milestones in the Expert Systems history. *(Adapted from Ref.1, p.11)*

As one can see, the Expert Systems history starts at around 1943 with the Post production rules. The initial trend was towards intelligent systems that relied on powerful methods of reasoning rather than making use of domain knowledge. They were also designed to cover *any* domain rather than a specialised area.

The shortcoming of this approach was that the machine had to *discover* everything from first principles in every new domain. In contrast, a human expert would rely on domain knowledge for high performance.

3.2 Milestones.

3.2.1 The Beginning.

It all started in the late 50s and early 60s, when several programs aimed at general problem solving were written. Post *(Ref 3)* was the first to use the *production systems* in symbolic logic, proving that any system of mathematics or logic could be written as a certain type of production rule system. The basic idea was that any mathematic /logic system is just a set of rules specifying how to change a string of symbols into another set of symbols.

The major difference between a Post production system and a human is that the manipulation of strings is only based on syntax and not on any semantics or understanding of the underlying knowledge. A Post production system consists of a group of production rules. The limitation of the Post production system is that, if two or more rules apply, there is no control specifying whether to apply one, some or *all* rules, and in what *order*. The Post production rules are *not* adequate for writing practical programs because of the lack of a *control strategy*.

At the same time (around 1943), McCulloch and Pitts developed a mathematical modelling of *neurons*, which later gave birth to the *connectionist* programming paradigm. Hebb gave an explanation of the learning process of the neurons in 1949.

The next step happened around 1954, with the introduction of a control structure for the Post production system, by Markov. A so-called *Markov algorithm* is an ordered group of productions, applied in order of priority to an input string. The termination of the Markov algorithm is triggered by either a production not applicable to the string or by the application of a rule terminated with a period. The Markov algorithm as a definite control strategy. The higher priority rules are used first. If not applicable, the algorithm tries the lower priority rules. Although applicable to a real expert system, the Markov algorithm may take a long time to execute on a system with many rules. If the response time is too long, the system will *not* be used no matter how good it is.

The possible improvement to Markov's approach of always trying *all* the rules, was of course to be able to apply *any* rule, in a non-sequential order. This would be possible if the algorithm already *knew* about all the rules. This was to happen later on, around 1977 - 1979.

3.2.2 The human problem solving model.

An outstanding example of a problem solving program was to be the *General Problem Solver*, described in the published work *Human Problem Solving* *(Ref 2)*, by Newell and Simon. Their work demonstrated that much of the human problem-solving (cognition) could be expressed as IF..THEN *production rules*.

Newell and Simon regarded the knowledge as *modular*, each rule corresponding to a small module of knowledge called a *chunk*. In this model, the human memory is organised chunks loosely arranged, with links to other related chunks. Sensory input would provide stimuli to the brain and trigger appropriate rules of *long term memory*, which then produce the suitable response. The *long term memory* is where the knowledge is stored; the *short term memory* however, is used for temporary knowledge storing during problem solving.

Human problem solving must also include a so-called *cognitive processor*. This is the equivalent of the *inference engine* in a modern expert system. The cognitive processor will try to find the rules to be activated by the appropriate stimuli. If more than one rule is to be activated, then a conflict resolution must be performed in order to decide which rule will be activated first (priority assignment).

Newell and Simon have not only managed to create a credible model of human problem solving, but have also set the basis of the modern rule-based expert systems. The concept of knowledge chunks brings with it the problem of *granularity* in the design of expert systems. *More* granularity means several chunks of knowledge are blended together in one rule; less granularity makes an isolated rule difficult to understand.

A. A change of concept.

The General Problem Solver reflected the current trend of the sixties to attempt producing general-purpose systems that heavily relied on reasoning and not on domain knowledge. Both approaches have their advantages and disadvantages though. Domain knowledge may be powerful, but it is *limited* to a particular domain. Applying it to another domain involves *skill* rather than genuine *expertise*.

In the early seventies it was already obvious that the only way to achieve machines comparable in performance with human counterparts was to use domain knowledge. In many cases, human experts rely less on reasoning and a lot more on a *vast* knowledge of heuristics (practical knowledge) and experience. When a human expert is confronted with a substantially different problem, he/she must use reasoning from the first principles and they are no better in this respect than any other person. Therefore, expert systems relying solely on reasoning are doomed to failure. The factors that contributed to the creation of the modern rule-based systems are presented in **Fig. 2. 2**.

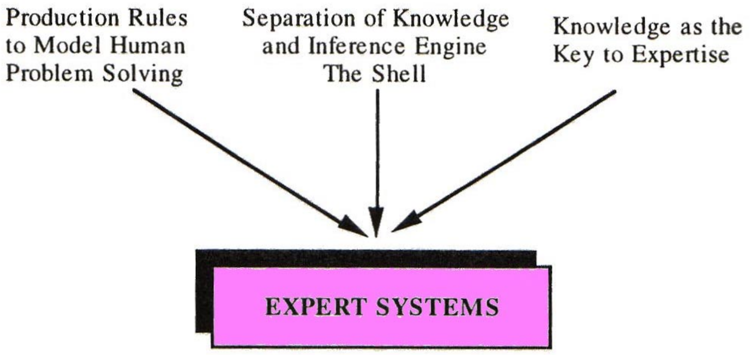

Production Rules to Model Human Problem Solving
Separation of Knowledge and Inference Engine The Shell
Knowledge as the Key to Expertise
EXPERT SYSTEMS

Fig. 2. 2 Creation of the modern rule-based expert system *(Ref.1, p.15)*

In 1965, work has begun on DENDRAL, the first expert system, aimed at identifying chemical constituents. However, DENDRAL was an unique system in which the knowledge base was mixed together with the inference engine. This made the task of reusing it to build a new expert system almost impossible.

1972-73 brought the development of the MYCIN system, intended to diagnose illnesses. MYCIN was a step forward because:

- it has demonstrated that AI may be used in real-life situations;
- MYCIN was used to test and prototype the *explanation facility*, the automatic knowledge acquisition (bypassing the explicit knowledge coding) and even intelligent tutoring (found in the GUIDON system);
- the concept of expert system *shell* was demonstrated to be viable.

MYCIN explicitly separated the knowledge base from the inference engine, making the core of the system *reusable*. Emptying out the knowledge base of the MYCIN system for example, and reusing the inference and explanation modules could therefore build a new expert system. The shell produced by removing the medical knowledge from the MYCIN system produced EMYCIN (Empty MYCIN).

Back to the improvements in the control algorithms. In 1979 Charles R. Forgy, at the Carnegie-Mellon University developed a fast pattern matcher algorithm, which stores information about rules in a network. The *RETE* algorithm is fast because it does *not* match facts against *every* rule on every *recognise-act* cycle. Instead, it looks for *changes* in matches. Because the static data does not change, it can be *ignored* and the processing speed increases dramatically. This algorithm was used to develop the OPS (Official Production System) language and shell.

The foundations of the modern rule-based expert systems are shown in **Fig. 2. 3**.

A variation of the DENDRAL system was developed in 1978, called META-DENDRAL. This program used *induction* to infer new rules of chemical structure. META-DENDRAL has been one of the most important attempts to defeat the *acquisition bottleneck*, where the knowledge has to be extracted from a human expert and explicitly coded into the knowledge database.

Another example of meta-reasoning was the TEIRESIAS knowledge acquisition program for MYCIN. In case of an erroneous diagnosis, TEIRESIAS would interactively lead an expert through the reasoning MYCIN has accomplished, until the point where the mistake was made is discovered. The acquired rule would first be tested for *compatibility* with the existing rules, and if no conflicts found, added to the knowledge base. Otherwise, TEIRESIAS would query the user about the conflict. Meta-knowledge in TEIRESIAS can be either a *control strategy* (on how the rules must be applied), or *validation-and-verification* meta-knowledge (whether the rule is in the correct form so it can to be added to the knowledge base).

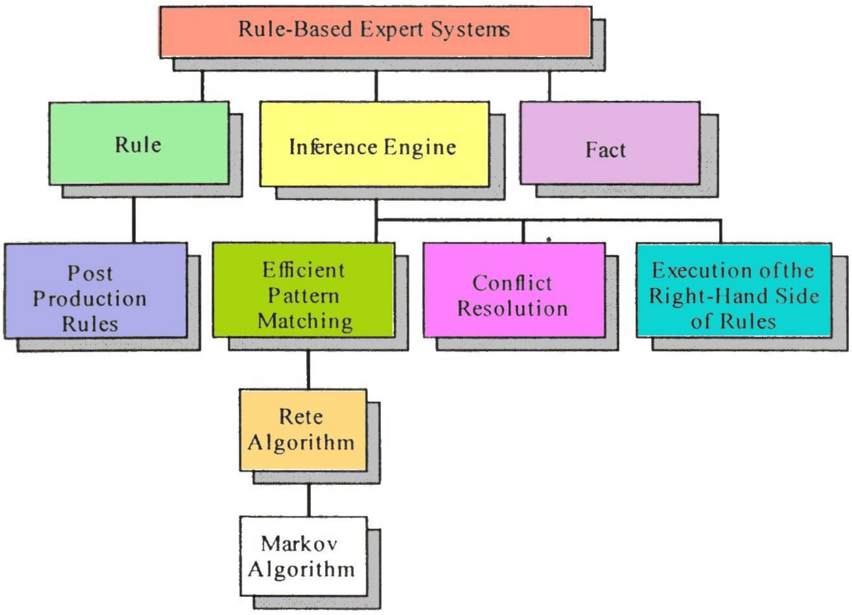


Fig. 2. 3 The foundation of the rule based expert systems *(Ref.1, p.32)*

3.2.3 A new trend.

Around 1980, companies start producing commercial expert systems, accompanied by powerful software for expert system development. An example is the Automated Reasoning Tool (ART, released later on in 1985). Also in 1980, new *specialised* hardware was developed, optimised to run the expert system software faster. The *LISP machines* were specially designed to run the most used expert system development base language of the period, LISP. The native assembly language, operating system, etc were actually created in LISP (LISt Processing language).

As one can imagine, this type of dedicated technology came to a price. While being more efficient and faster at running LISP code, the dedicated machines had price tags around $100000. Obviously there was a need for more affordable software that could run on common machines.

1985 is another milestone to the development of expert systems, with the introduction of *CLIPS* by NASA. Written in C, CLIPS is portable (i.e. suitable for a large number of platforms), fast (it uses the *RETE* pattern matching algorithm developed by Forgy in 1979) and cheap or free for some users. CLIPS can run on any machine that can run the standards C language. CLIPS had actually made expert system development and study accessible to large groups of potential users. This is always vital in promoting the acceptance of new technologies.

1990 sees the revival of the neural networks with the Artificial Neural Systems (ANS, refer section E, part 1). In 1992, 11 shell programs were available for Macintosh, 29 for MS-DOS programs, 4 for UNIX platforms and 12 for dedicated mainframe applications *(Ref.7)*.

By 1994, commercial ANS are being offered (refer section E, part 3) by various companies at different levels of price and features. The number of expert systems patents in the US has grown steadily from 20 in 1998 to over 900 in 1996.*(Ref.7)*.

The GUESS system (Generically Used Expert Scheduling System) , by Liebovitz et al, was devised in 1997, using an object oriented, constraint-based AI toolkit approach to perform scheduling.

The current (1997-) trend cannot escape the influence of the Internet. Envisaged applications are:

- intelligent customer and vendor agents for use on the web in E-commerce;
- intelligent search engines;
- *interagent* communications methods relating to AI in E-commerce.

3.2.4 Artificial Neural Systems

3.2.4.1 Basic concepts.

Artificial Neural Systems (ANS) have tried to solve problems by modelling the human brain information processing. They represent a different programming paradigm, also known as *connectionism*. Connectionism models solutions to problems by training simulated neurons connected in a network. ANS may well be the ideal front end to expert systems that require massive amounts of sensor inputs and fast response.

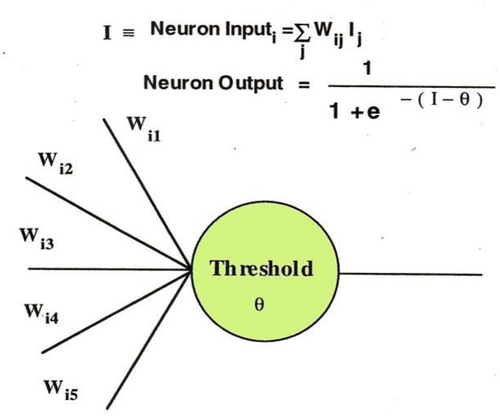

$I \equiv$ Neuron Input$_i = \sum_j W_{ij} I_j$
Neuron Output $= \frac{1}{1 + e^{-(I-\theta)}}$
W_{i1}
W_{i2}
W_{i3}
W_{i4}
W_{i5}
Threshold
θ

Fig. 2. 4 A neuron processing element *(Ref.1, p.45)*

ANS uses simple processing elements connected in parallel, called *neurons*. A neuron may have multiple inputs but only one output. The key to the functioning of an ANS is the *weights* associated with each element (refer **Fig. 2. 4**). Each element (neuron) has an associated *threshold* value θ, subtracted from the input I. The input signals of the neuron are multiplied by the weights and summed to yield the total neuron input, equal to 1. The output is given by the formula in **Fig. 2. 4** and it is called an *activation function*.

3.2.4.2 ANS advantages.

Since ANSes are modelled after current brain theories, information is represented by *weights*. The distributed representation of information is similar to a *hologram*, where the lines of the hologram diffract a laser beam passing through in order to reconstruct the stored image.

Advantages of ANS over conventional computers:

- fault tolerant storage. Removing portions of the net will only determine a degradation of the quality in the stored data;
- graceful degradation of the quality of the stored image, in proportion to the amount of net removed;
- data is stored in the form of *associative memory* (i.e. where partial data is sufficient to recall the complete stored information);
- nets can *extrapolate* and *interpolate* from their stored information. A net can be trained to seek significant features / relationships in the data. Subsequently, the net may suggest relationships with the new data.
- nets display *plasticity*. After the removal of a part of the neurons in the net, the net can be retrained to its original skill level if sufficient neurons are left;
- ANSes are simple to build and cheap to maintain, mainly because of plasticity. However, for number intensive tasks or algorithmic solutions, ANSes are *not* a good choice.

3.2.4.3 ANS timeline.

The origin of artificial neural systems dates back to 1943 (refer paragraph A). However, a milestone in the connectionist AI approach was 1961, with Rosenblatt's definition of *perceptron*, a new type of artificial neuron. The perceptron consisted of two layers of neurons, and a basic learning algorithm. It displayed remarkable capabilities for learning and pattern recognition. However in his model, the weights had to be set manually.

In 1969, a new milestone occurs with the publication of the book *Perceptrons* by Minksy and Papert. The book showed the limitations of the perceptrons as general computing machines, proving that a perceptron could not recognise the exclusive OR logic function. The book has had a negative impact on the ANS research. Eventually, new methods of representing symbolic AI information by frames became popular in the 1970s, but research only continued on a small scale.

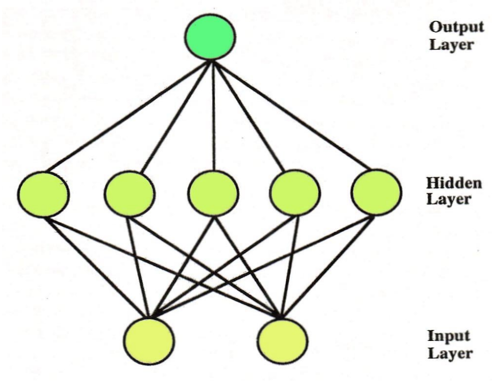


Fig. 2. 5 A back propagation net *(Ref.1, p.46)*

In 1982 though, Hopfield's work brought a revival of ANS by introducing the *two-layer Hopfield Net* and demonstrating that ANS *could* solve a wide variety of problems. The *back-propagation* net, commonly implemented as a three-layer net, is an example of ANS that can solve the XOR problem (refer **Fig. 2. 5**).

Other types of ANS emerged, such as the *counter-propagation* net, invented by Hecht-Nielsen in 1986. It was also proven that a three-layer network with *n* inputs and *2n+1* neurons in the hidden layer could map *any* continuous function.

The present trend in expert systems *is* towards the artificial neural nets. Also, there are masses of commercially available expert systems and shells.

New companies and existing firms have been organised to develop ANS technology and products. For example, NestorWrite is a program by Nestor Corp., which can recognise handwritten input and convert it to text.

ANS simulators and hardware accelerator boards (to speed up the learning process) are also marketed by a number of companies such as Texas Instruments, Synaptics, Neural Tech, etc.

3.2.5 The future.

The future of expert systems is bright. Expert systems are being already used in almost all aspects of our life, from space travel to agriculture, Internet to underwater devices. Wherever there is a need to solve ill-structured problems in real-time, to be 99.9% reliable and unemotional and to have an unlimited capacity for learning and remembering, the expert systems will do nicely.

Artificial Intelligence itself is advancing rapidly, and with the latest progress in genetics, the first human druids may not be that far away in time. Such developments may create totally new social and ethical problems, which may also need to be investigated.

4 Knowledge Representation in Expert Systems.

This chapter will try to present the meaning of knowledge and some commonly used representations of knowledge for expert systems.

4.1 What *is* knowledge ?

Knowledge is a very hard to define term and it has many meanings. *Data*, *facts*, *information* are terms also being used with the meaning of knowledge.

The study of knowledge is *epistemology*. It contains:

- philosophic theories (Aristotle, Plato, Descartes, Kant, etc)
- *a priori* knowledge (considered to be universally true);
- *a posteriori* knowledge (derived from the senses).

Knowledge may be classified in:

- procedural knowledge (*know-how)*;
- declarative knowledge (declarative statements);
- tacit knowledge (*unconscious knowledge*) - cannot be expressed by language. A good example is *how to contract a muscle*.

Knowledge is of central importance to expert systems. **Fig. 3. 1** shows a possible hierarchy of knowledge. The of this structure foundation is *noise*, meaning items of little interest and obscure data. Items of potential interest make up the next level, the *data*. Processed data represents *information*, and sits on top of the *data*. The *knowledge* layer, consisting of specialised information, is next. The last and most interesting layer is the *meta-knowledge*, (*meta* meaning *above*).

Meta knowledge is knowledge about knowledge. For example in an expert system with knowledge in several domains, the meta-knowledge would specify *which* knowledge is applicable for a particular domain. Within a particular domain, meta-knowledge may be used to determine *which* set of rules is applicable for a specific problem.

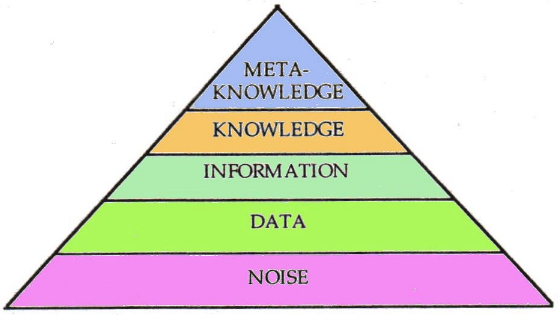


Fig. 3. 1 The hierarchy of knowledge *(Ref.1, p.59)*

4.2 Knowledge representation techniques.

4.2.1 Productions.

Production rules are commonly used in the knowledge bases of expert systems. A formal notation for defining productions is the Backus-Naur form (BNF). BNF represents *metalanguage* for defining the *syntax* of a language. While syntax defines the form, the *semantics* refers to the meaning of the language. A grammar is a complete set of production rules that uniquely defines a language.

There is a wide variety of languages that BNF can define, such as natural, logic, mathematical, computer.

4.2.2 Semantic Nets.

Semantic network (nets) are also known as *propositional* nets. A proposition is a statement that can either be true or false, and is a form of *declarative* knowledge (they state facts).

4.2.2.1 Concepts.

The structure of a semantic net may be shown graphically in terms of nodes (*objects*) and arcs (*links* or *edges*) connecting them. Nodes represent objects, concepts or situations. The links show the relationships between nodes. If the links are directed arrows, then the net becomes a *directed graph* (refer **Fig. 3. 2**). Relationships give the knowledge contained in the nodes a cohesive structure about which other knowledge may be *inferred*.

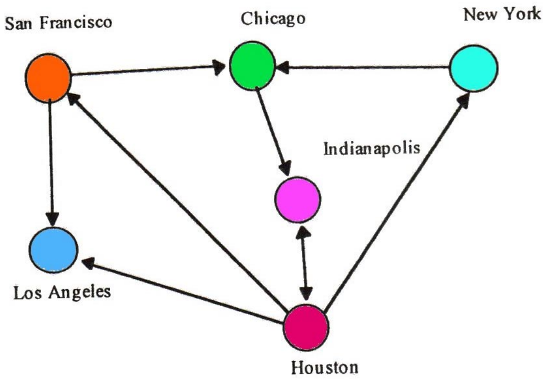


Fig. 3. 2 A type of semantic net (directed graph) *(Ref.1, p.64)*

4.2.2.2 Semantic net problems.

Unfortunately, semantic nets have some limitations in representing knowledge:

- Lack of link name standards and naming rules for the nodes. Only if the link and nodes are unambiguously defined may the semantic net represent *definitive* knowledge (knowledge that can be defined).
- Semantic nets cannot define knowledge in a similar way to logic;
- Heuristics (rules of thumb) on how to efficiently search the net *cannot* be embedded in a semantic net. Heuristics play a *major* role in expert systems.

Although a number of approaches have been tried to correct the above-mentioned problems, little improvement has been achieved at the expense of the natural expressiveness of the semantic nets.

4.2.3 Schemata.

All the knowledge in a semantic net is contained in the links and nodes. Therefore, a semantic net is an example of *shallow knowledge structure*. In contrast, a *deep* knowledge structure may explain why something occurs via *causal* knowledge. In a human expert's case, deep knowledge is usually called upon when causal knowledge fails to solve the problem.

Schemas are used to represent more complex knowledge structures. Conceptual schemas are *abstractions* in which specific objects are classified by their general properties. Schemas are used to focus on the *general* properties of an object without being distracted by irrelevant details.

Schemas have an internal structure to their nodes, in contrast with the semantic nets. A semantic net is like a data structure in computer science. The schema is more like a data structure in which nodes contain records that can further contain records, data or pointers to other nodes.

4.2.4 Frames.

Two types of schemas have been used in many AI applications: the *frame* and the *script* (a time-ordered sequence of frames)

Frames are very suitable for representing objects typical to a particular situation (stereotypes). *Commonsense* knowledge is very difficult to master by the computers, and frames can be of great help. While semantic nets are used to represent broad knowledge, frames are efficient at representing a narrow subject containing much *default* knowledge. Special purpose language have been designed for frames, such as FRL, SRL, KRL, etc

An analogy can be made between a frame and a *record* in Pascal or an *atom* in LISP. The correspondent frame elements to the fields and values of the record are the *slots* and slot *fillers* of a frame. The fillers may be values (e.g. a property) or a *range* of values (a *type* slot). Slots may contain attached procedures which can be *if-needed* (when a filler value is needed but none present), *default* (expectations of a situation), or *if-added* (when a value is added to a slot).

Sophisticated frame systems have been used in discovering mathematical concepts and describing mathematical understanding in linear algebra.

Frame systems that allow unrestricted alteration or cancellation of slots may however display major problems. Most frame systems do not provide a way to define unchangeable slots. This leads to a system where nothing is really certain and no universal statements can be made. Building composite elements from simpler frames is also restricted.

4.2.5 Logic and Sets

Knowledge may also be represented by symbols of *logic* (the study of exact reasoning). Logic is extremely important in expert systems, as the inference engine reasons from facts to conclusions. The term of *automated reasoning systems* would include both logic programming and expert systems.

Two common formal logic methods of representing knowledge are:

- *syllogisms*. Invented by Aristotle in the fourth century B.C., they have two *premises* and one *conclusion*, which is inferred from the premises. The premises provide the evidence from which the conclusion must follow. The classical example is *(Ref.1, p.64)*:
    - o *Premise:* All men are mortal
    - o *Premise:* Socrates is a man
    - o *Conclusion:* Socrates is mortal
- *Venn Diagrams*. In this model, a circle represents a *set*, and the objects contained in the set are the *elements* (refer **Fig. 3. 3**). Socrates is an element, therefore is represented as a dot and not a circle.

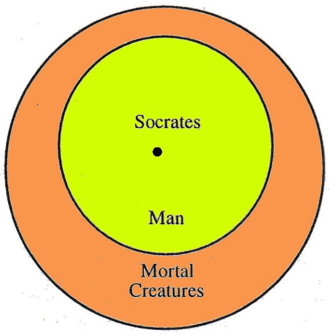


Fig. 3. 3 A Venn diagram for the example *(Adapted from Ref.1, p.79)*

The detailed descriptions of the knowledge representation methods previously presented are beyond the scope of this paper. The interested reader is encouraged to refer to *Ref.1, pp. 57-91*.

5 Conclusions.

This paper aimed to achieve two objectives:

- give a basic coverage on some important areas of expert systems, and maybe incite the reader into further individual expert systems technology reading and research;
- allow the author to train its own *natural* neural net (pun intended) with knowledge in the field of AI and expert systems.

The history of Artificial Intelligence and expert systems started in fact very early, with the ancient philosophers and then Renaissance scientists. The foundations of modern expert systems are quite recent though (i.e. 1943 onwards). In terms of *computer history* however, this is equivalent to hundreds of years of human history.

While Artificial Intelligence is far from being perfect and expert systems still have a long way to go to fully model a human expert, the progress achieved in such a short period is astonishing. And better still, the development pace is accelerating constantly, in step with computer hardware technology.

⇧ Back to Table Of Contents.

6 References.

1. Giarratano & Riley (1998) *Expert Systems: Principles and Programming*, 3rd Ed., PWS Publishing Company, Boston, MA, ISBN 0 534 95053 1
2. Allen Newell and Herbert A. Simon, (1972) *Human Problem Solving*, Prentice-Hall.
3. Emil L. Post (1943), *Formal Reductions of the General Combinatorial Decision Problem*, American Journal of Mathematics
4. George F. Luger, William A. Stubblefield (1993) *Artificial Intelligence - Structures and Strategies for Complex Problem Solving*, Benjamin-Cummings, Albuquerque, ISBN 0-8053-4780-1
5. Stephen L. Gallant, *Connectionist Expert Systems*, Comm of the ACM, Feb 1998
6. Buchanan, Feigenbaum, *Dendral and META-Dendral: Their Applications Dimension,* Artificial Intelligence, 1978.
7. *AI Magazine*, *IEEE Journa*l, *AI Expert* publications, *AAAI* web site papers.

⇧ Back to Table Of Contents.