[Please cite as:
Forsyth, R.S. (1990). Chapter 1: developments in artificial intelligence. In: Mirzai, A.R. (ed.) *Artificial Intelligence: Concepts and Applications in Engineering*. London: Chapman & Hall Ltd. pp. 3-34.
]

[**NOTE**: The small-scale examples of knowledge-based systems shown herein are merely illustrative: the Grand National rule-base is no longer valid, entry conditions of that race having changed markedly since this chapter was written, and the Derby rules were never profitable. *Caveat aleator*!]

Chapter 1
Developments in Artificial Intelligence

R. Forsyth

1. Historical Review

Mary Shelley's deranged doctor, Victor Frankenstein, is taken by many to be the fictional archetype of the AI scientist [29]. Psychiatrically speaking, his character was the portrayal of a manic depressive: he laboured long and feverishly to create his monster; but when he saw the results of his work, fell into despondency and torpor.

Thus it is highly appropriate that, since the invention of the digital computer in the mid-1940s, the discipline of AI has been characterized by a similar cycle of mania followed by depression. There have been several bouts of excessive enthusiasm (as one generation of workers overturned the ideas of their predecessors) followed by disillusionment (as the limitations of the new approach became clear).

In fact it is possible to identify five boom-then-bust episodes in the history of AI which, for the sake of simplicity, I have divided into five decade-long chunks in Table 1.

This summarizes the "mainstream" of AI development, though, of course, it should be remembered that AI researchers do not change the whole thrust of their work on 31 December in every tenth year. It should also be remembered that there are in every phase a handful of mavericks working outside the main conceptual framework -- either using a previously discarded approach or one whose time has yet to come (which, as we shall see, sometimes amounts to the same thing).

Decade	Label	Main Concern
1950s	The Dark Ages	Neural Networks
1960s	The Age of Reason	Automated Logic
1970s	The Romantic Movement	Knowledge Engineering
1980s	The Enlightenment	Machine Learning
1990s	The Gothic Revival	Neural Nets Revisited

Table 1.1 -- Short history of AI.

1.1 The Dark Ages: Black Boxes and Grey Matter

Almost as soon as it was devised, the digital computer was seen as a natural vehicle for the exploration of intelligence. In the 1950s computers were routinely referred to as "electronic brains"; and two of the visionaries most responsible for the design of the digital computer, John von

Neumann and Alan Turing, speculated on the possibility of simulating, and indeed surpassing, human intelligence with machines [31,32].

The phrase "artificial intelligence" was not coined till the Dartmouth Summer School of 1956, so the field was initially part of cybernetics. During this period the central idea was that the way to make machines intelligent was to mimic the brain. Workers took a bottom-up approach, assuming that putting together a large network of artificial neurons and subjecting it to an appropriate training schedule was the best way to build intelligent systems. In other words they thought that if they could model the brain, a mind would emerge as an inevitable consequence.

This approach freed researchers from having to analyze high-level mental processes (like thinking, remembering, understanding speech and so on). But it had two serious disadvantages: (1) the hardware of the time simply was not capable of supporting sufficiently large networks of artificial neurons to give rise to anything resembling human intelligence; and (2) the physiology of the brain was, and still is, very imperfectly understood.

Thus by the early 1960s the neural-simulation approach had fallen into disrepute but not before some interesting systems had been built. We briefly consider just two of them: Pandemonium and the Perceptron.

Pandemonium [27,28] was a pattern-recognition system based on a four-level computational architecture. At the bottom level were data demons which sampled basic attributes of the input pattern, such as whether a patch in an image was light or dark. Each data demon was linked to one or more computing demons at the next level up. These computed simple functions (such as product, difference, maximum) of their inputs and then passed their output up to the next level, the cognitive demons. There was one cognitive demon for every different type of pattern that the system could be taught to recognize. Thus if the task was to discriminate between written digits (0 to 9) there would be ten cognitive demons (See Figure 1.1).

Each cognitive demon calculated the weighted sum of the inputs it received from all the computing demons and "shrieked" with an intensity proportional to this sum at the top-level demon, the decision demon of which there was only one. The decision demon simply picked the cognitive demon with the "loudest shriek" as identifying the class of the input pattern.

Figure 1.1 -- Pandemonium.

Pandemonium could learn in two ways: (1) by adjusting the weights attached to the connections between computing and cognitive demons; (2) by replacing, every so often, a few computing demons having very small weights with new ones obtained by "mutated fission" and "conjugation".

Although now of purely historical interest, Pandemonium incorporated some very important ideas. Firstly, it was explicitly, though loosely, based on theories about the nervous system, and thus represents one of the earliest connectionist models. Secondly, it employed what would now be termed a genetic algorithm [13] in its method of replacing ineffective computing subdemons by mutation and recombination of more useful ones. Thirdly, it introduced a precursor of fuzzy logic [35], in that the logical and relational operations performed by computing demons gave smoothly varying outputs. For instance, a subdemon that computed the truth value D1 > D2 had a logistic curve as its output rather than a step function; that is, it was scaled according to the size of the difference D1 - D2. Fourthly, and most importantly, Pandemonium was a learning machine. Its designer, Selfridge, like most of his contemporaries, assumed that intelligence could not be pre-programmed but would have to be taught. This is an insight which has recently been rediscovered.

Despite incorporating so many modern ideas, and despite Selfridge's picturesque language, Pandemonium had less effect on the course of AI than a slightly later and slightly simpler neural-net model, the Perceptron invented by Frank Rosenblatt [22,23]. (See Figure 1.2)

The original Perceptron was an optical pattern classification device. Its input layer was a grid of 400 photocells which corresponded (very roughly) to the retina of the vertebrate eye. The photocells responded with 1 for light and 0 for dark. They were randomly connected to the next layer, the associator units, which computed a threshold function: if an associator unit's total input exceeded its threshold its output was 1, otherwise 0. Thus the associators were simple feature detectors or pattern primitives rather like Pandemonium's computing demons.

All the associators were linked to the final layer, containing one processing element for each type of pattern. (In the simple case of only two patterns, a single processing element, with a Yes/No output, was sufficient.)

The final processing element took a weighted sum of the values on its input lines and compared it to a threshold: if the sum exceeded the threshold it signalled that its specific pattern was present, otherwise that it was absent. The way the system learned was by alteration of the weights.

Image matrix   Association units   Weighting factors   Summation unit

Figure 1.2 -- A Perceptron.

Association units combine single pixel responses into group features. Each feature is weighted and the sum is compared to a threshold to make the final class decision.

Rosenblatt's main achievement was to devise a training strategy for Perceptrons and to prove that it would converge on an optimal set of weights provided that the input patterns were "linearly separable", which means that they could be partitioned into hypercube-shaped regions in feature space. The following error-correction algorithm was applied after every training presentation.

1. If the output is correct, leave the weights unaltered.
2. If the output should have been 1 but was 0, increment the weights on the active input lines (i.e. those with values above 0).
3. If the output should have been 0 but was 1, decrement the weights on the active input lines.

The increment could be fixed, but it was more usual for it to be proportional to the difference between the weighted sum and the desired output.

Perceptrons were trained to perform useful pattern-recognition tasks, and variants of the Perceptron learning system [33] have been built into most telecommunication lines for adaptive noise reduction over the last twenty years; but in the end the AI community grew dissatisfied with this approach. Learning the difference between linearly separable classes of patterns ceased to seem very exciting. Most AI workers drew the conclusion (somewhat prematurely) that neural models would always be limited to emulating very low levels of intelligence.

1.2 The Age of Reason: Logic and Language

In the 1960s, as it became apparent that building brains was not a viable AI strategy, a new fashion took over. This was a top-down approach, according to which computers, humans and other creatures were treated as species of information processor. The mechanisms of the nervous system were no longer a concern, since a program can be executed on many different physical systems without reference to their underlying hardware.

The chief proponents of this view were Allen Newell and Herbert Simon of Carnegie-Mellon University [20]. They argued that the correct level of description for human problem solving was in terms of comparing, storing and manipulating symbols. They were interested in the kind of rational behaviour in which a problem solver takes a symbolic description of a problem situation and systematically transforms the elements of that description by applying formal rules in a sequence of steps to achieve a new state -- namely, the solution. They hoped to find broad general principles of intelligence which would be common to all forms of rational problem solving, from business decision-making to playing games, and which could equally well be carried out by humans or by computers.

This approach succeeded with formal puzzles and games, but tended to break down when applied to the kind of open-ended, ill-structured problems which people deal with in everyday life. Two systems that exemplify both its strengths and weaknesses were GPS (the General Problem Solver) and SHRDLU.

GPS [5] was given a task environment for a particular problem in terms of routines for matching pairs of objects, routines for applying operators to objects and routines to determine whether an operator could be applied to an object. These objects were symbol structures, which could also be considered as states or situations.

GPS went to work by setting goals, detecting differences between the current state and its goal, and then finding operators that would reduce those differences -- a technique known as means-end analysis. Each problem generated subproblems till a subproblem was reached that could be solved in one step. The system proceeded by successive solution of its subproblems till it achieved its goal or gave up.

GPS had three main methods: (1) transforming one state into another; (2) reducing the difference between two states; and (3) applying an operator to an object or state. Methods could call each other -- including themselves -- as subprocesses. Endless repetition was avoided by maintaining a list of states already encountered. GPS also used a technique, called "planning", which abstracted from two objects A and B new objects A' and B' by eliminating most of the details. It then attempted to turn A' into B' (typically an easier task than turning A into B) and if successful used the simplified solution to guide work on the full problem.

The idea of means-end analysis was taken further in STRIPS, the Stanford Research Institute Planning System [6], where it was used for robotic planning. Here the objective was for a robot to find its way around in a household environment.

Operators in STRIPS were defined in terms of preconditions and effects. The preconditions were those facts which had to be true to apply the operator. The effects were facts that would become false (the Delete-list) or become true (the Add-list) when the action was carried out. For example the operator GO-INTO-ROOM might be defined as follows.

```
GO-INTO-ROOM (self, Room) :

Preconditions : at(self,P); outside(P,Room);
near(P, D); door(D); connects(D,Room,P) ; not shut(D).
```

```
Add-List : in(self,Room).
Delete-List : at(self,P).
```

If the robot had a goal of entering a particular room (which might itself be a subgoal in a higher level plan) it would look for an operator which had being in a room in its Add-list, such as GO-INTO-ROOM. It would then try to satisfy the preconditions of that operator. With the above example, it might well find that all preconditions were true except that the door was not open. This would cause it to seek operators that had opening the door on their Add-lists. In the end this kind of reasoning could lead to quite long sequences of purposeful behaviour, involving, for instance, finding a key, bringing it to the door, turning it in the lock, twisting the door handle, pushing the door, and so on.

STRIPS was further developed into ABSTRIPS [25] but by that time this line of research was running out of steam. Planning based on search through a propositionally encoded state space seemed incredibly pedantic when compared to what people do. It was logical and it worked (within limits), but it failed to capture the essence of human problem-solving.

SHRDLU [34] was a rather different kind of system which could take part in a conversation, via keyboard and screen, about a simple world containing children's building blocks. This blocks-world was also represented on the screen and contained a simulated robot arm, to which the user could give orders in natural English, producing dialogues like the following (where machine replies are in capitals).

Pick up a big red block.
OK.
Grasp the Pyramid.
I DON'T UNDERSTAND WHICH PYRAMID YOU MEAN.
Find a block which is taller than the one you are holding and put it into the box.
BY 'IT' I ASSUME YOU MEAN THE BLOCK WHICH IS TALLER THAN THE ONE I AM HOLDING. OK.
What does the box contain?
THE BLUE PYRAMID AND THE BLUE BLOCK.

Winograd's work introduced a number of new ideas about the relationship between syntax and semantics and about representing linguistic information in a computer, but -- though it made a great impact at the time -- it did not lead to a breakthrough in machine understanding of unrestricted natural language.

By the early 1970s, the GPS approach, based on heuristic search of a state space, was seen to have run into a dead end. Though GPS could solve cryptarithmetic and other puzzles, there was no known way of extending it to deal with the kind of fuzzy, ill-defined problems which regularly crop up in economics, medicine, meteorology and a wide range of other domains. Likewise, it was realized, when the initial euphoria had worn off, that Winograd's SHRDLU could not easily be "scaled up" from its micro-world of children's blocks to cope with the full diversity of natural language.

1.3 The Romantic Movement: Knowledge as Power

The key to the next phase of AI was the abandonment of the goal of general intelligence. Observing that human experts are competent in their field by virtue of a large body of specialist knowledge, a team led by Edward Feigenbaum at Stanford University began developing the first expert systems. This coincided with a move of the global centre of gravity in AI from the east to the west coast of the USA -- roughly speaking, from MIT to Stanford.

The expert or knowledge-based system is a caricature of the human expert, in the sense that it knows almost everything about almost nothing. For example, an expert system for blood infections may not even have any knowledge about lung diseases, let alone about another area like currency trading. None the less, this narrowing of focus enabled expert-system designers to achieve impressive practical results. Among the most noteworthy systems developed during this period were DENDRAL, MYCIN and Prospector.

DENDRAL [2] was concerned with the interpretation of data obtained from a mass spectrometer. This instrument bombards a chemical sample with a beam of electrons, causing its molecules to break up. The fragments are passed through a magnetic field which deflects those with a high charge and low mass more than those with a low charge and large mass. As a result, a plot showing the relative abundance of fragments with different mass-to-charge ratios is produced.

Such plots give the trained analytical chemist an insight into the structure of the compound under test. The trouble is that a complex molecule can fragment in many ways, i.e. that different chemical bonds can be broken inside the mass spectrometer. Whereas it is relatively easy to predict that the cleavage of a specific chemical bond will give rise to a peak in the plot at a particular point, it is much harder, working in reverse, to take the plot with many different peaks and work out what chemical decomposition produced it. In other words, there is a problem of inverse mapping.

DENDRAL helped to solve this problem by using a constrained generator which, given a molecular formula, proposed a number of ways it could be structured. With a complex compound there might be literally millions of potential structures. DENDRAL used fragmentation rules, rules of chemistry and the data from the mass spectrometer to eliminate the vast majority of these, leaving a small candidate set for the human chemist to consider.

A further development, Meta-DENDRAL, employed a learning algorithm to refine and extend DENDRAL's rule-set. It has been reported [18] that Meta-DENDRAL made genuinely novel contributions to the science of organic chemistry.

DENDRAL was the first knowledge-based system but two other pioneering expert systems, MYCIN and Prospector, have been even more influential in the development of the field. MYCIN [30] was a system for diagnosing bacterial blood infections and prescribing appropriate drug therapy. Prospector [9] was a system for interpreting the results of geological investigations, which became famous after assisting in the discovery of large molybdenum deposits in the USA and Canada.

All three systems, though very different in application, shared some important common features which have become hallmarks of the expert system. Above all, they imposed a division between knowledge and reasoning, so that an expert system today is seen to be composed of two core parts: the knowledge base and the inference engine.

The knowledge base contains symbolic representations of experts' rules of judgement in a format that permits the inference engine to perform deductions on it. The inference engine consists of whatever search and reasoning procedures are necessary to deploy that knowledge -- in other words to draw conclusions from it.

Expert systems like MYCIN brought many new ideas into AI, and into mainstream computing, but perhaps the most significant was this notion of explicit representation of knowledge and its separation from the inference process.

1.4 The Enlightenment: Learning how to Learn

AI is still living, to a large extent, off the intellectual capital accumulated in the expert-system boom of the 1970s. But during the 1980s expert systems as such have ceased to be the focal point of AI research. At the cutting edge of AI research in the 1980s has been the issue of machine learning, which is a major theme of the present book.

This is not so much because expert systems have failed as because they have succeeded. The idea of representing knowledge explicitly -- as rules, frames or in other forms -- has stimulated many new developments that would not otherwise have taken place. But it has also become widely recognized that knowledge elicitation and codification is a difficult and labour-intensive task [12,15]. It is therefore quite natural to think in terms of automating this knowledge acquisition process, or at least assisting it by machine. This brings us back to the problem of machine learning (see also [8]).

Successful learning programs have been developed by Michalski [17] and by Ross Quinlan [21], but these will be covered in chapter 3 of the present volume; so here we consider two other, contrasting, learning systems: EURISKO and WISARD.

EURISKO [16] was a development of an earlier system called AM [4] which explored the domain of elementary number theory. EURISKO was intended to extend AM's applicability to other domains than mathematics and to remedy some of the weaknesses which prevented AM going further than it did. It was tested in several domains: elementary mathematics (where it reproduced the discoveries of AM), designing VLSI components (where it invented a three-dimensional logic gate), and the design of battle fleets in a naval wargame called the Trillion Credit Squadron competition. In this last context, EURISKO twice won the US national championship with bizarre fleets which, essentially, exploited previously unnoticed loopholes in the rules.

EURISKO worked according to an agenda, which it could manipulate itself. Tasks on the agenda were given numeric worths, from 0 to 999, which determined how much processing time could be devoted to them. When it finished one task, EURISKO would pick the next task to work on by selecting the one with the highest worth rating. While working on a task, other tasks could be added to the agenda, or could have their worths increased or decreased. One of the interesting things about EURISKO was that it never really needed to stop: it could go on introspecting for ever -- creating new tasks and revising its conceptual vocabulary until someone halted it. (What generally happened, however, was that a point of diminishing returns was reached where even the highest-scoring tasks on the agenda had low values.)

Not only could EURISKO create new concepts, by applying heuristic rules to existing concepts, it could also devise new heuristics. This was possible since rules, concepts and practically everything else in the system were represented in a uniform language known as RLL-1 (Representation Language Language 1). RLL-1 is a sophisticated object-oriented language.

Thus EURISKO could start out with a rule such as,

```
        IF a rule has fired only once
        THEN try to generalize it.
```

Treating this rule as data rather than program, it might find that the rule itself had fired only once. Then it could be applied to itself, to generate

```
        IF a rule has fired less than 4 times
```

THEN try to generalize it.

Possibly this rule would turn out to be more useful than its parent.

Despite EURISKO's self-reflexive capacity, it did not go further in elementary mathematics than AM; and for the time being its inventor, Lenat, has turned aside from discovery programs altogether. He is currently working on a different project called CYC which aims to computerize a very large knowledge base by encoding a household encyclopaedia and adding all the unstated, 'common sense' knowledge needed to interpret it. This is a long-term project, but when it is completed Lenat believes there will at last be sufficient background knowledge for computerized discovery systems -- perhaps quite similar to EURISKO in overall plan -- to go beyond the barriers that held up AM and EURISKO just when they were getting really interesting. Till this 'world lexicon' is ready we will not know for sure whether EURISKO got stuck through lack of background knowledge or because of intrinsic defects in its mode of operation (See also [11]).

Another innovative learning system, of a completely different type, was the WISARD visual pattern recognizer [1]. The interesting point about WISARD is that it was a parallel computer with no processors, only memory. (See Figure 1.3)

Image data digitalized as ones and zeroes
Address generated -- 177
Octuple

Figure 1.3 -- WISARD schematic.

Groups of eight pixels are arbitrarily connected to eight-bit registers to form the basis of WISARD's pattern recognition system. Each octuple corresponds to a bank of 256 memory locations. During the training phase the value in the octuple forms an address within its RAM bank dependent on the pixel pattern and a 1 is written to this address. If the same pattern is present during the recognition phase then the same RAM bank address will be generated and the 1 present will signify recognition.

It worked by inspecting a TV image digitized as 512 x 512 pixels, which were sampled in groups (normally groups of 8). Each 8-bit grouping was a random sample of 8 pixels in the image, which could be in one of 256 states, since pixels have only two values, 0 or 1. In effect these 8-bit groupings were feature detectors, and the current state of each one said something about the picture currently on view.

Each feature detector was connected to its own private bank of 256 RAM locations, using a content-addressing scheme. The state of the detector (an 8-bit value) pointed to one of its 256 RAM locations. Prior to training, all the bits were set to zero. During the training phase, the bit addressed by every detector would be set to 1 if the image being presented was a positive example of the concept to be learned, or left alone if not. There were actually 32768 detectors, requiring a total of one megabyte of RAM (not a huge memory these days).

After training, when a new image was shown to the system, each detector addressed a location within its RAM bank (as in the training phase) and retrieved the bit stored there -- 0 or 1. These were just added up: a high total indicated that many detectors were in the same state as when a positive training instance had been displayed; hence that the present image was also likely to be a positive case. A low total suggested the opposite. (Extension to deal with more than two classes merely required more memory.)

WISARD, in effect, computed a 'fingerprint' of what it saw and compared this with past experience. In practice it proved highly resistant to distortions in the image. For example, it was taught to discriminate between smiling and frowning faces. This is just the sort of thing that computers, programmed in the traditional manner, have hitherto been strikingly poor at doing.

EURISKO and WISARD between them exemplify two divergent approaches to the problem of machine learning -- a problem that will have to be addressed if artificial intelligence is ever to become a reality (since intelligence without learning is scarcely conceivable). At present the AI community seems to favour the route pioneered by WISARD rather than that followed by EURISKO.

1.5 The Gothic Revival

As we enter the 1990s a counter-reformation is underway within AI. Research students and research grants are being diverted away from the symbolic knowledge-based paradigm which held sway in the 1970s and 1980s towards neuro-computing or "connectionism". I call this AI's "Gothic revival" because we have now come full circle. The irony is that neural-net computing was the guiding theme of the 1950s.

Once more, extravagant claims are being made about the potential of the new (or newly rediscovered) technology, and the race is on, especially in the US and Japan, to become world leader in neuro-computing.

Why should the neural-net approach succeed now when its limitations were exposed more than 25 years ago? After all, though there has been progress in neurophysiology, there has been no tremendous breakthrough in our understanding of the brain during the intervening years.

The answer is twofold. In the first place, modern computing machinery is many orders of magnitude more powerful anything available to the cyberneticians of the 1950s, and computer power is still increasing relentlessly. It is possible to construct neural nets with hundreds of thousands of processing elements which, while not rivalling the capabilities of higher mammals, do approach or even exceed the complexity of the nervous systems of insects, molluscs and other 'simple' creatures.

Secondly, and perhaps more importantly, learning rules for multi-layered neural networks have been discovered. A particular weakness of the Perceptron -- identified by Minsky and Papert [19] when they delivered the *coup de grace* to earlier work on neural networks -- was the fact that it had only one layer of internal processing elements. This was not because multi-layer systems have no advantages over single-layer systems (they have) but because Rosenblatt's error-correcting rule could not be guaranteed to converge on a solution when used with multi-layered systems.

In the 1980s a number of workers independently discovered what has become known as the back-propagation rule, which overcomes this difficulty. Perhaps the most notable success of back propagation is Sejnowski's NETtalk [26], a text-reading system. (See figure 1.4)

NETtalk scans seven characters at a time, and attempts to pronounce the middle item of the seven. When it has done so, the text is moved along and it does the same for the next character -- again in the context of three other characters on either side. With 309 processing units and about 18000 weights, NETtalk was trained using a back-propagation algorithm on several thousand words of text. It progressed from meaningless babble to 98% correct pronunciation in 16 hours. This contrasts dramatically with the performance of DECtalk, a rule-based system built by Digital Equipment Corporation which initially inspired NETtalk: DECtalk is said to have required well over 20 man-years to encapsulate the knowledge of phonetics and English spelling needed for correct pronunciation.

Output to speech synthesizer chip
26 phonetic units
80 units in 'Hidden' Layer
7 x 29 = 203 lnput Units

Figure 1.4 -- Outline diagram of NETtalk.

Back propagation computes an error term at the output layer (typically half the sum of the squared differences between the actual output values and the desired target values) and feeds this back through the network [24]. Provided that the transfer function of the processing units in the internal layers can be differentiated, the error signal can be fed back all the way to the input units, adjusting connection weights as it goes. This is a simple method of credit apportionment which does for multi-stage neural networks (which are not limited to linearly separable pattern classes) what Rosenblatt's error-correction procedure did for the Perceptron.

With a sigmoid transfer function (quite commonly used in practice), the error term (de) for each internal or 'hidden' processing cell (j) at a given level (s) is computed as

$$de(j,s) = o(j,s) \times (1 - o(j,s)) \times wsum(j,s + 1)$$

where o(j,s) is the output from unit j at level s and wsum(j,n) is the product of weight times error-term summed for all units at level n that receive output from unit j on level n-1, (n = s+1). Then the weight change (dw) to be added to all weights at a given level (s) is

$$dw(j,s) = lc \times de(j,s) \times o(j,s - 1)$$

where lc is an (adjustable) learning coefficient. Although back propagation is easy to grasp, intuitively satisfying and has become almost an "industry standard", it is not a very efficient training rule. More sophisticated weight-adjustment algorithms do exist. (See chapter 4 of this volume.)

2. Expert Systems in Practice

However, we have been anticipating events somewhat. Although the leading edge of AI has moved on to machine learning, and more recently to neural-net models of learning, the state of the art in applied AI is still the expert system. Knowledge engineering is flourishing, and the immediate future will involve progress in the techniques used in knowledge engineering.

In the early 1980s, commercial expert-system vendors tended to re-create a classic expert system like MYCIN or Prospector, remove the domain-specific knowledge, and release the resulting inference engine, together with some kind of rule editor, as an expert-system shell. Today that is no longer sufficient: users realize that for large-scale applications additional facilities are required, such as screen-handling, access to existing databases, procedural code and so on. Over the last four or five years, expert-system shells have evolved by acquiring extra facilities so that the more modern products are, in fact, complete software development environments. Thus the distinction between expert-system shells and AI programming environments is becoming increasingly blurred. Two commercial products that illustrate this trend are Leonardo [3] and Egeria [14]. They are used below to illustrate, with small-scale examples, the kind of work that is being done in this field.

2.1 Grand National Betting Advisor

The first example was put together with the aid of some old horse-racing yearbooks. The Grand National -- the most popular single betting event in Britain -- is widely regarded as a lottery. But in fact there is enough regularity in the results over the years to make it an interesting (if sometimes frustrating) exercise in prediction.

For example, although the race is known as a 'graveyard for favourites' it is still usually won by a well-backed horse. To be quantitative, just under 40% of the winners from 1946 to 1988 were returned at odds of 10/1 or less, while 65% have been returned at odds of 20/1 or less. Yet in a typical field fewer than 10% of the horses will have odds of 10/1 or less, while only 25% will have odds at or below 20/1. Thus 65% of the winners come from 25% of the runners, indicating that the betting is a reasonable guide to a horse's prospects.

Another reliable indicator is the weight carried. An obvious thought is that the lighter the load which a horse is required to carry round 4.5 miles and over 30 fences the better. But historically horses carrying heavier weights have done well. To be specific, of the last 29 races 24 have been won by a horse carrying more than ten stone; yet in an average race only just over a third of the runners carry more than ten stone. Thus 83% of the winners come from 35% of the horses. The explanation for this seemingly puzzling fact is that the race is a handicap, so the better horses are given heavier loads. As there is a minimum weight, many of the animals carrying ten stone (the minimum weight) are carrying a greater burden than their form would warrant: they are actually overloaded.

It is also a good sign when the horse has won at least one of its last four outings. If so, as you would expect, it has a better chance of being placed than otherwise.

In addition, there has been a tendency, over the last 12 races, for 9, 10 and 11 year-olds to do better in terms of proportion finishing in the first four than other age groups. Presumably around ten years there is some kind of equine athletic peak (though this effect is weaker than those already mentioned).

Such, then, is the reasoning behind the rule-base expressed below in the Leonardo rule language.

```
grandnat                                                                                  9-Mar-89
------------------------------------------------------------------------------------------------------------

/* Rule-base for the Grand National (Bayesian) :
/* by R.S. Forsyth, April 1988

control bayes

control 'threshold 0.02'

seek backability

if start is yes then use introscreen; intro is done

if intro is done
and horsename is not '????'
and won_in_last_4 > 0 { Ls 1.88 Ln 0.6 }
then backability is ok { prior 0.1 }
```

```

if age_of_horse > 8
and age_of_horse < 12 { Ls 1.2 Ln 0.7 }
then backability is ok { prior 0.1 I

if forecast_odds < 20 { Ls 2.5 Ln 0.4g }
then backability is ok

if weight_carried is 'more than 10 stone' {Ls 2.36 Ln 0.44 }
then backability is ok

if going is not heavy { Ls 1.25 Ln 0.8 }
then backability is ok

/* heavy going tends to produce funny results.

/* Preamble is in IntroScreen
/* Postscript is in Conclusion slot of Backability
```

There is not space here to explain the system in depth, but it should be noted that this example uses Leonardo's Bayesian inference method, which requires values for LS (Logical Sufficiency) and LN (Logical Necessity) to be specified.

The goal of the system is to establish a value for the variable BACKABILITY -- which can be interprered as the probability of the horse under consideration finishing in the first four (thus gaining at least place-money). The prior probability of this happening is 0.1, since the race limit is 40 entrants and it is hardly ever under-subscribed.

Leonardo, like similar systems, prompts for values of variables having a bearing on the goal to be sought (such as AGE-OF-HORSE) from the user; and then uses Bayes's rule to update the probability of BACKABILITY. The apparent simplicity of the rule-base is slightly deceptive, as there are other elements that do not appear in these rules (e.g. an introductory message). Of these background elements the most important is a little procedure that decides on acceptable odds once BACKABILITY is known, listed below.

```
Name: Vfmodds
Longname:
Type: Procedure
AcceptsText: back
LocalReal: cf, vfm
Body:

screen(63)
global
cf = certainty(back) / 400
vfm = ((1-cf)/cf)
vfm = int(vfm+0.5)
box (5,10,10,70,31,1)
at(7,15, horsename,' should be worth an each-way bet if')
at(8,15, 'you can get odds of ',vfm as 'xxx',' to 1')
```

This is Leonardo's rather roundabout way of defining a procedure called VFMODDS, which is used at the end of the consultation to tell the user what odds would represent value for money. The details are not very important here, but to understand what is going on you need to know that Leonardo is an object-oriented system. Rules implicitly declare objects, such as GOING and BACKABILITY. Each object is described by a frame that consists of a number of slots (or attributes), such as

> Name: age-of-horse
> Value: 8
> Certainty: {1.0}

and many more besides. Some slots are filled by Leonardo with default values; others are optional. It so happens that an object of type Procedure must have a slot called

> Body:

which contains statements in Leonardo's procedural programming language. This allows a knowledge base to contain a mixture of declarative and procedural code.

VFMODDS is actually called by putting the line

> Conclusion: {run vfmodds(backability)}

as a slot in BACKABILTTY's frame. This is an optional slot that says what should happen when the system gets to a conclusion about an object's value.

2.2 Derby Day

Our second example concentrates on another major sporting occasion, also the medium of heavy betting, the Epsom Derby. Here we illustrate a different software package, Egeria. In Egeria there is no division, as there is in Leonardo, between the procedural programming language and the rule language.

The system listed below is the result of an afternoon spent poring over some racing annuals for the years 1983 to 1990. It is a simple betting advisor for the Epsom Derby. It won't make your fortune, but it will almost certainly save you from backing some real no-hopers, especially if you use it over a number of years.

It rates any horse according to three attributes which are fairly easy to find out about. (Some other possible attributes were considered, but they had little effect.) These are:

> Odds: Forecast betting odds
> Lastrace: Finishing position in horse's last race
> WonTrial: Whether horse has won a recognized trial:
>> Sandown trial
>> Newmarket 2000 Guineas
>> Lingfield trial
>> York Dante stakes.

The program applies the rule

```
If odds >= 20 or lastrace > 3 then 'Hopeless'
else if odds < 10 and lastrace = 1 then 'Probable'
else if WonTrial then 'Probable'
else 'Possible'
```

to split the runners into three categories. As a matter of interest, the results for each of those categories over the last 8 runnings of the race (1983-90) are tabulated below.

	Winners/Runners	Placed (1,2,3)/Runners
Probable	6/30	14/30
Possible	2/28	7/28
Hopeless	0/74	3/74

Table 1.2 -- Derby Results, 1983-1990.

Essentially this is a way of eliminating over half the contestants, which have very little chance of winning. A few refinements to the bare rule have been added in the Egeria program.

(Readers may find it amusing to try out either or both of these knowledge bases in practice. N.B. Please don't expect miracles!)

/* .... Egeria program removed, system no longer extant; rules above should suffice for those keen to gamble .... */

Once again the details are not so important as the general principles, though it is worth noting the difference between tasks and procedures. This example program contains one of each, a task (Main) and a procedure (Advisor). The task calls the procedure.

Procedures are called at specific points in the program -- just as in conventional programming languages like C or Pascal. Tasks, on the other hand, are executed when their triggering condition becomes true. This provides a kind of event-driven programming. In the present case, however, this flexibility is not utilized: Main is simply triggered off when the program begins.

Like Leonardo, Egeria offers:

- object-oriented structures (with class inheritance)
- uncertainty management, in various forms
- procedural code
- multiple alternative representation methods

Between them, Egeria and Leonardo give a good idea of the sort of facilities available for developing knowledge-based systems in a modern knowledge-engineering software environment.

3. Towards the Sixth Generation

The knowledge bases listed in the previous section are unrealistic in size, but realistic in being hand-crafted. That is to say someone (the present author in this case) had to think up the rules. This is one of the weak points of expert systems, and one of the reasons why machine learning is enjoying a revival [7]. If we want truly intelligent machines the issue of machine learning will have to be tackled, and there are signs on the horizon that the effort needed to do so has already begun.

3.1 Beyond 1992

In 1992 the Japanese Fifth Generation project, which caused great excitement in the computer industry when it was launched a decade earlier, reaches its conclusion. It is already safe to state that it will fail to attain some of its declared objectives. Goals likely to remain unrealized by the 1992 deadline include:

(1) unrestricted machine translation
(2) applied speech understanding
(3) image understanding systems
(4) highly parallel general-purpose inference engines.

These, and some other aims, would -- we were initially led to believe -- come to fruition within the time span of the ten-year project. But their realization has been postponed.

In Europe (and probably North America) such an outcome would no doubt lead to the discrediting of a whole line of research -- i.e. to another paradigm shift within AI. The Japanese, however, are not responding in that way: they do not plan to abandon what they have done so far. Instead they intend to build on it by trying harder (with, admittedly, a change of emphasis).

The Fifth Generation project will almost certainly be followed by a sixth Generation initiative, which will pick up many of the unfulfilled aims of the Fifth Generation as well as adding new goals of its own. The shape of the sixth Generation is becoming clear, at least in outline, and it is certain that it will be a dominant influence on the course of AI from now till the end of the twentieth century. Just as the centre of AI moved west across America in the 1970s, so now we are witnessing its move, further west, across the pacific.

The Sixth Generation (6G) is a rather tenuous concept, perhaps even more so than the Fifth Generation before it. But it is a useful term, none the less, for gathering together several disparate strands at the leading edge of AI research. According to Gomi [10], the phrase was first seriously used in a 1985 report from a working group at the Japanese Agency of Science and Technology (JAST). However the Ministry of International Trade and Industry forced JAST to stop using such terminology and in fact suppressed that report, fearing that it would have a negative impact on the Fifth Generation project, which then had more than half its life still to run.

Today, however, many Japanese (and some Americans) are starting to talk more freely about 6G computing and what it would entail. Two groups in particular have taken up the ideas embodied in the lost JAST report, and embellished them some more. These are the Human Frontier science Program (HFSP) researchers and the Advanced Telecommunications Research (ATR) group. Their plans for the 6G explicitly call for major advances in

- psychology
- physiology
- linguistics
- logic

as well as computer science and AI. Applications will include:

- machine translation
- intelligent autonomous robots
- real-time expert systems

- analogical and qualitative reasoning

Two specific projects, above all, typify the challenge which these research groups expect to meet:

(1) computerized cars;
(2) translating telephones.

3.2 Autonomous Automobiles

At present it would be quite absurd to expect a computer-controlled vehicle to travel for more than 25 meters down a busy city road without crashing. Driving is one of those skills that has so far resisted all attempts at automation. Several teams in the USA, largely under military direction, have been working on Autonomous Land Vehicle (ALV) projects, but even allowing for military secrecy they do not appear to have succeeded.

At HFSP (roughly speaking, Japan's answer to the Stanford Research Institute) they believe that the failure of previous efforts to build an ALV can be explained, at least in part, by over reliance on symbolic processing. By integrating the perceptual and learning abilities of large-scale neural nets as subsystems within a modular cognitive architecture, they hope to create, by early next century, an ALV which will not cause panic and danger when let loose in urban traffic.

A successful ALV demands the solution of a high proportion of the outstanding problems at the frontiers of AI. Among other things, it would have to be able to do real-time image processing (possibly supplemented by sonar or radar information); it would have to integrate long-term and short-term planning in a dynamic environment; and, above all, it would have to be adaptive, since its plans would need amendment from minute to minute to deal with the unexpected. The challenge posed by designing an ALV leads us to reflect that a typical truck driver exhibits skills much further beyond the current boundaries of AI than those of, say, a chess grandmaster.

Of course, we should not idealize human abilities: humans are not very good at driving either. In 1987 in the UK, 69418 people were seriously injured in road accidents (of whom 5125 died). Thus there were, on average, 190 serious injuries every day that year on British roads. More concretely, a quick survey outside my house prior to writing this sentence revealed evidence of collision damage on 3 of the 7 vehicles parked in a quiet side street. This is only a tiny sample, but it does show that accidents are relatively normal. That is one reason why robotic transport might be desirable.

3.3 Translating Telephones

Perhaps even more ambitious are the plans of ATR. They have a budget of 615 million US dollars for the first 7.5 years of a 15-year advanced research programme which began in 1986. According to Dr Koichi Yamashita, president of ATR's Kansai Science City laboratory, their main goal is research into what they call HOCS (Human Oriented Communications Systems). A central part of this programme is the development of a telephone system that will understand utterances by parties in foreign languages (e.g. English) and translate them -- in real time -- into Japanese. Their plan calls for the building of a prototype system, operating between Japan and the United States, by 2001. Less complete prototypes are to be tested earlier, including a major milestone demonstration in 1995.

The system will have to handle continuous speech, from many different speakers, with a large vocabulary, in real time. In addition, it is foreseen that most of the input will be ungrammatical: people do not actually talk in well-formed sentences. Allowing irregular input will result in a high degree of uncertainty in the translation process, making accurate translation extremely difficult.

ATR plans to use an interlingual representation, such that all foreign language input will be translated into a common internal base (not Japanese but a kind of universal semantic code) which will then be rendered into Japanese, or possibly other languages. This goes well beyond anything currently possible, or even planned, elsewhere in the world.

Even if this particular project is a "glorious failure" it is very probable that its technological spin-off will include a number of inventions that will confirm Japan's status as world leader in telecommunications. And the scientific knowledge gained on the way could lead to a revolution in linguistics.

3.4 A Prognostication

The idea of a telephone exchange that understands enough about human concerns and human communication to perform simultaneous translation on multiple channels is, strictly speaking, fantastic. (Surely an intellect capable of such a feat would be driven insane by being buried beneath Tokyo and compelled to listen to incessant chatter?) I personally would be most surprised if such a device is created within the time scale envisaged. But that does not mean that it is a futile enterprise. On the contrary, the translating telephone, like the ALV, is an excellent focus for 6G research.

In conclusion: the 6G is still little more than a collection of buzz words, but it would be wrong to dismiss it on that account. There is a well-funded, worldwide effort in progress to make a reality of what until recently would have been seen as science fiction. The Japanese, especially, are determined to make the 6G more than just a slogan.

If, by the early years of the 21st century, ordinary folk really can pick up a phone in California and speak fluent English to a monolingual Japanese speaker (and be understood without perceptible delay) or can hail a ride in a computer-driven taxi, then AI will at last have earned its name; and a profound change in the whole character of civilization will have taken place.

If such things do not come to pass, AI will once again have promised more than it can deliver, and the search for a new paradigm will begin all over again.

References

[1] I. Aleksander & P. Burnett, *Reinventing Man*, Pelican Books, Middlesex, 1984.

[2] B. Buchanan & E. Feigenbaum, "DENDRAL and Meta-DENDRAL: their Applications Dimension", *Artificial Intelligence*, Vol-11, pp 5-24, 1978.

[3] Creative Logic Ltd, *Leonardo: the Manual*, Creative Logic Ltd, Uxbridge, 1988.

[4] R. Davis & D. Lenat, *Knowledge based Systems in Artificial Intelligence*, McGraw-Hill, New York, 1982.

[5] G.W. Ernst & A. Newell, *GPS: a Case Study in Generality and Problem Solving*, Academic Press, New York, 1969.

[6] R. Fikes & N. Nilsson, "STRIPS: a New Approach to the Application of Theorem Proving to Problem Solving", *Artificial Intelligence* , Vol-2, pp 189-208 , 1971.

[7] R. Forsyth, *Machine Learning: Principles and Techniques*, Chapman & Hall, London, 1989.

[8] R. Forsyth & R. Rada, *Machine Learning*, Ellis Horwood, Chichester, 1986.

[9] J. Gaschnig, "Prospector: an Expert System for Mineral Exploration", in *Introductory Readings in Expert Systems*, D. Michie (ed.), Gordon & Breach, New York, 1982.

[10] T. Gomi, "Sixth Generation Computing by the year 2001", *Proc. 2nd International Symposium on Artificial Intelligence and Expert Systems*, BCEEP, Berlin, pp 21,3-312, 1988.

[11] K. Haase, "Automated Discovery", in *Machine Learning: Principles and Techniques*, R. Forsyth (ed.), Chapman & Hall, London, 1989.

[12] A. Hart, *Knowledge Acquisition for Expert Systems*, Kogan Page, London, 1986.

[13] J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[14] ISI Ltd, *Expert Systems with Egeria*, Intelligent Systems International Ltd, Redhill, 1989.

[15] A. Kidd, *Knowledge Acquisition for Expert Systems*, Plenum Press, New York, 1987.

[16] D. Lenat, "Eurisko: a Program that learns new Heuristics and Domain Concepts", *Artificial Intelligence*, Vol-21, pp 61-98, 1983.

[17] R. Michalski & R.L. Chilausky, "Knowledge Acquisition by Encoding Expert Rules versus Computer Induction from Examples: a Case Study Involving Soybean Pathology", *Int. J. Man-Machine Studies*, Vol-12, pp 63-87, 1980.

[18] D. Michie & R. Johnston, *The Creative Computer*, Pelican Books, Middlesex, 1985.

[19] M. Minsky &. S. Papert, *Perceptons: an Introduction to Computational Geometry*, MIT Press, Cambridge Mass, 1969.

[20] A. Newell & H. Simon, *Human Problem Solving*, Prentice-Hall, Englewood Cliffs, 1972.

[21] J.R. Quinlan, "Induction of Decision Trees", *Machine Learning*, Vol-1, pp 81-106, 1986.

[22] F. Rosenblatt, "The Perceptron, a Perceiving and Recognizing Automaton", *Cornell Aeronautical Lab Report* 85-460-1, Cornell University, New York, 1957.

[23] F. Rosenblatt, *Principles of Neurodynamics*, Spartan Books, New York, 1962.

[24] D. Rumelhart & J. McClelland (eds.), *Parallel Distributed Processing*, Vols-1, & 2, MIT Press, Cambridge, Mass, 1989.

[25] E. Sacerdoti, "Planning in a Hierarchy of Abstraction Spaces", *Artificial Intelligence*, Vol-5, pp 115-135, 1,974.

[26] T. Sejnowksi & C.R. Rosenberg, "Parallel Networks that Learn to Pronounce English Text", *Complex Systems*, Vol-l, pp 145-168, 1987.

[27] O. selfridge, "Pattern Recognition and Modern Computers", *Proc . Western Joint Computer Conference*, Los Angeles, March 1955.

[28] O. Selfridge, "Pandemonium: a Paradigm for Learning", in *Mechanization of Thought Processes, HMSO*, London, 1959.

[29] M. Shelley, *Frankenstein: or, the Modern Prometheus*, Bantam Books, New York, 1981.

[30] E. Shortliffe, *Computer-based Medical Consultations: MYCIN*, Elsevier, New York, 1976.

[31] A. Turing, "Computing Machinery and Intelligence", *Mind*, 59, 256, pp 433-460, 1950.

[32] J. Von Neumann, *The Computer and the Brain*, Yale University Press, New Haven, 1958.

[33] B. Widrow & M. Hoff, "Adaptive Switching circuits*", Inst. Radio Engineers, Western Electronics Convention Record*, part 4, pp 96-104, 1960.

[34] T. Winograd, *Understanding Natural Language*, Edinburgh University Press, Edinburgh, 1972.

[35] L. Zadeh, "Fuzzy Sets", *Information and Control*, Vol-9, pp 338-353, 1965.