Expert system

From Wikipedia, the free encyclopedia

An **expert system** is software that uses a knowledge base of human expertise for problem solving, or clarify uncertainties where normally one or more human experts would need to be consulted. Expert systems are most common in a specific problem domain, and are a traditional application and/or subfield of artificial intelligence (AI). A wide variety of methods can be used to simulate the performance of the expert; however, common to most or all are: 1) the creation of a knowledge base which uses some knowledge representation structure to capture the knowledge of the Subject Matter Expert (SME); 2) a process of gathering that knowledge from the SME and codifying it according to the structure, which is called knowledge engineering; and 3) once the system is developed, it is placed in the same real world problem solving situation as the human SME, typically as an aid to human workers or as a supplement to some information system. Expert systems may or may not have learning components.

Expert systems were introduced by researchers in the Stanford Heuristic Programming Project, including the "father of expert systems" Edward Feigenbaum, with the Dendral and Mycin systems. Principal contributors to the technology were Bruce Buchanan, Edward Shortliffe, Randall Davis, William vanMelle, Carli Scott, and others at Stanford. Expert systems were among the first truly successful forms of AI software.[1][2][3][4][5][6]

The topic of expert systems also has connections to general systems theory, operations research, business process reengineering, and various topics in applied mathematics and management science.

Contents

Aspects

Certainty factors

The MYCIN rule-based expert system introduced a quasi-probabilistic approach called certainty factors, whose rationale is explained below.

A human, when reasoning, does not always make statements with 100% confidence: he might venture, "If Fritz is green, then he is probably a frog" (after all, he might be a chameleon). This type of reasoning can be imitated using numeric values called *confidences*. For example, if it is known that Fritz is green, it might be concluded with 0.85 confidence that he is a frog; or, if it is known that he is a frog, it might be concluded with 0.95 confidence that he hops. These certainty factor (CF) numbers quantify uncertainty in the degree to which the available evidence supports a hypothesis. They represent a degree of confirmation, and are not probabilities in a Bayesian sense. The CF calculus, developed by Shortliffe & Buchanan, increases or decreases the CF associated with a hypothesis as each new piece of evidence becomes available. It can be mapped to a probability update, although degrees of confirmation are not expected to obey the laws of probability. It is important to note, for example, that evidence for hypothesis H may have nothing to contribute to the degree to which Not_h is confirmed or disconfirmed (e.g., although a fever lends some support to a diagnosis of infection, fever does not disconfirm alternative hypotheses) and that the sum of CFs of many competing hypotheses may be greater than one (i.e., many hypotheses may be well confirmed based on available evidence).

The CF approach to a rule-based expert system design does not have a widespread following, in part because of the difficulty of meaningfully assigning CFs a priori. (The above example of green creatures being likely to be frogs is excessively naive.) Alternative approaches to quasi-probabilistic reasoning in expert systems involve fuzzy logic, which has a firmer mathematical foundation. Also, rule-engine shells such as

Drools and Jess do not support probability manipulation: they use an alternative mechanism called salience, which is used to prioritize the order of evaluation of activated rules.

In certain areas, as in the tax-advice scenarios discussed below, probabilistic approaches are not acceptable. For instance, a 95% probability of being correct means a 5% probability of being wrong. The rules that are defined in such systems have no exceptions: they are only a means of achieving software flexibility when external circumstances change frequently. Because rules are stored as data, the core software does not need to be rebuilt each time changes to federal and state tax codes are announced.

Chaining

Two methods of reasoning when using inference rules are forward chaining and backward chaining.

Forward chaining starts with the data available and uses the inference rules to extract more data until a desired goal is reached. An inference engine using forward chaining searches the inference rules until it finds one in which the if clause is known to be true. It then concludes the *then* clause and adds this information to its data. It continues to do this until a goal is reached. Because the data available determines which inference rules are used, this method is also classified as *data driven*.

Backward chaining starts with a list of goals and works backwards to see if there is data which will allow it to conclude any of these goals. An inference engine using backward chaining would search the inference rules until it finds one which has a *then* clause that matches a desired goal. If the *if* clause of that inference rule is not known to be true, then it is added to the list of goals. For example, suppose a rule base contains

1. (1) IF X is green THEN X is a frog. (Confidence Factor: +1%)
2. (2) IF X is NOT green THEN X is NOT a frog. (Confidence Factor: +99%)
3. (3) IF X is a frog THEN X hops. (Confidence Factor: +50%)
4. (4) IF X is NOT a frog THEN X does NOT hop. (Confidence Factor +50%)

Suppose a goal is to conclude that Fritz hops. Let X = "Fritz". The rule base would be searched and rule (3) would be selected because its conclusion (the *then* clause) matches the goal. It is not known that Fritz is a frog, so this "if" statement is added to the goal list. The rule base is again searched and this time rule (1) is selected because its then clause matches the new goal just added to the list. This time, the *if* clause (Fritz is green) is known to be true and the goal that Fritz hops is concluded. Because the list of goals determines which rules are selected and used, this method is called *goal driven*.

However, note that if we use confidence factors in even a simplistic fashion - for example, by multiplying them together as if they were like soft probabilities - we get a result that is known with a confidence factor of only one-half of 1%. (This is by multiplying 0.5 x 0.01 = 0.005). This is useful, because without confidence factors, we might erroneously conclude with certainty that a sea turtle named Fritz hops just by virtue of being green. In Classical logic or Aristotelian term logic systems, there are no probabilities or confidence factors; all facts are regarded as certain. An ancient example from Aristotle states, "Socrates is a man. All men are mortal. Thus Socrates is mortal."

In real world applications, few facts are known with absolute certainty and the opposite of a given statement may be more likely to be true ("Green things in the pet store are not frogs, with the probability or confidence factor of 99% in my pet store survey"). Thus it is often useful when building such systems to try and prove both the goal and the opposite of a given goal to see which is more likely.

Software architecture

The following general points about expert systems and their architecture have been outlined:

1. The sequence of steps taken to reach a conclusion is dynamically synthesized with each new case. The sequence is not explicitly programmed at the time that the system is built.

2. Expert systems can process multiple values for any problem parameter. This permits more than one line of reasoning to be pursued and the results of incomplete (not fully determined) reasoning to be presented.

3. Problem solving is accomplished by applying specific knowledge rather than specific technique. This is a key idea in expert systems technology. It reflects the belief that human experts do not process their knowledge differently from others, but they do possess different knowledge. With this philosophy, when one finds that their expert system does not produce the desired results, work begins to expand the knowledge base, not to re-program the procedures.

There are various expert systems in which a rulebase and an inference engine cooperate to simulate the reasoning process that a human expert pursues in analyzing a problem and arriving at a conclusion. In these systems, in order to simulate the human reasoning process, a vast amount of knowledge needs to be stored in the knowledge base. Generally, the knowledge base of such an expert system consists of a relatively large number of "if/then" type statements that are interrelated in a manner that, in theory at least, resembles the sequence of mental steps that are involved in the human reasoning process.[7]

Because of the need for large storage capacities and related programs to store the rulebase, most expert systems have, in the past, been run only on large information handling systems. Recently, the storage capacity of personal computers has increased to a point to which it is becoming possible to consider running some types of simple expert systems on personal computers.[7]

In some applications of expert systems, the nature of the application and the amount of stored information necessary to simulate the human reasoning process for that application is too vast to store in the active memory of a computer. In other applications of expert systems, the nature of the application is such that not all of the information is always needed in the reasoning process. An example of this latter type of application would be the use of an expert system to diagnose a data processing system comprising many separate components, some of which are optional. When that type of expert system employs a single integrated rulebase to diagnose the minimum system configuration of the data processing system, much of the rulebase is not required since many of the optional components will not be present in the system. Nevertheless, early expert systems required the entire rulebase to be stored since all the rules were, in effect, chained or linked together by the structure of the rulebase.[7]

When the rulebase is segmented, preferably into contextual segments or units, it is then possible to eliminate the portions of the rulebase containing data or knowledge that is not needed in a particular application. The segmentation of the rulebase also allows the expert system to be run on or with systems having much smaller memory capacities than was possible with earlier arrangements, since each segment of the rulebase can be paged into and out of the system as needed. Segmentation into contextual units requires that the expert system manage various intersegment relationships as segments are paged into and out of memory during the execution of the program. Since the system permits a rulebase segment to be called and executed at any time during the processing of the first rulebase, provisions must be made to store the data that has accumulated up to that point so that later in the process, when the system returns to the first segment, it can proceed from the last point or rule node that was processed. Also, provisions must be made so that data that has been collected by the system up to that point can be passed onto the second segment of the rulebase after it has been paged into the system, and data collected during the processing of the second segment can be passed to the first segment when the system returns to complete processing that segment.[7]

The user interface and the procedure interface are two important functions in the information collection process.

End user

There are two styles of user-interface design followed by expert systems. In the original style of user interaction, the software takes the end-user through an interactive dialog. In the following example, a backward-chaining system seeks to determine a set of restaurants to recommend:

> Q. Do you know which restaurant you want to go to?
> A. No
>
> Q. Is there any kind of food you would particularly like?
> A. No
>
> Q. Do you like spicy food?
> A. No
>
> Q. Do you usually drink wine with meals?
> A. Yes
>
> Q. When you drink wine, is it French wine?
> A. Yes

The system must function in the presence of partial information, since the user may choose not to respond to every question. There is no fixed control structure: Dialogues are dynamically synthesized from the "goal" of the system, the contents of the knowledge base, and the user's responses.

This approach wastes much of the user's time, because it does not allow a priori volunteering of information that the user considers important. In the previous example, the user must cycle through the entire series of questions rather than simply providing that they are looking for a moderately-priced Northern Italian, French or Turkish restaurant with a large wine selection, not more than 20 minutes driving distance. Therefore this approach is unlikely to be acceptable to busy users (e.g. a mobile-device user who needs to obtain information as efficiently as possible). Consequently, it has fallen into disfavor. Commercially viable systems will try to optimize the user experience by presenting options for commonly requested information based on a history of previous queries of the system using technology such as forms, augmented by keyword-based search. The gathered information may be verified by a confirmation step (e.g., to recover from spelling mistakes), and now act as an input into a forward-chaining engine. If confirmatory questions are asked in a subsequent phase, based on the rules activated by the obtained information, they are more likely to be specific and relevant.

In an expert system, implementing the ability to learn from a stored history of its previous use involves employing technologies considerably different from that of rule engines, and is considerably more challenging from a software-engineering perspective. It can, however, make the difference between commercial success and failure. A large part of the revulsion that users felt towards Microsoft's Office Assistant was due to the extreme naivete of its rules ("It looks like you are typing a letter: would you like help?") and its failure to adapt to the user's level of expertise over time (e.g. a user who regularly uses features such as Styles, Outline view, Table of Contents or cross-references is unlikely to be a beginner who needs help writing a letter).

Explanation system

Another major distinction between expert systems and traditional systems is illustrated by the following answer given by the system when the user answers a question with another question, "Why", as occurred in the above example. The answer is:

> A. I am trying to determine the type of restaurant to suggest. So far Indian is not a likely choice. It is possible that French is a likely choice. If I know that if the diner is a wine drinker, and the preferred wine is French, then there is strong evidence that the restaurant choice should include French.

It is very difficult to implement a general explanation system (answering questions like "Why" and "How") in a traditional computer program. An expert system can generate an explanation by retracing the steps of its reasoning. The response of the expert system to the question "Why" exposes the underlying knowledge structure. It is a rule; a set of antecedent conditions which, if true, allow the assertion of a consequent. The rule references values, and tests them against various constraints or asserts constraints onto them. This, in fact, is a significant part of the knowledge structure. There are values, which may be associated with some organizing entity. For example, the individual diner is an entity with various attributes (values) including whether they drink wine and the kind of wine. There are also rules, which associate the currently known values of some attributes with assertions that can be made about other attributes. It is the orderly processing of these rules that dictates the dialogue itself.

Comparison to problem-solving systems

The principal distinction between expert systems and traditional problem solving programs is the way in which the problem related expertise is coded. In traditional applications, problem-related expertise is encoded in both program and data structures. In the expert system approach all of the problem expertise is encoded *mostly* in data structures.

An example, related to tax advice, contrasts the traditional problem solving program with the expert system approach. In the traditional approach, data structures describe the taxpayer and tax tables, while a program contains rules (encoding expert knowledge) that relate information about the taxpayer to tax table choices. In the expert system approach, the latter information is also encoded in data structures. The collective data structures are called the knowledge base. The program (inference engine) of an expert system is relatively independent of the problem domain (taxes) and processes the rules without regard to the problem area they describe.

This organization has several benefits:

- New rules can be added to the knowledge base or altered without needing to rebuild the program. This allows changes to be made rapidly to a system (e.g., after it has been shipped to its customers, to accommodate very recent changes in state or federal tax codes).
- Rules are arguably easier for (non-programmer) domain experts to create and modify than writing code. Commercial rule engines typically come with editors that allow rule creation/modification through a graphical user interface, which also performs actions such as consistency and redundancy checks.

Modern rule engines allow a hybrid approach: some allow rules to be "compiled" into a form that is more efficiently machine-executable. Also, for efficiency concerns, rule engines allow rules to be defined more expressively and concisely by allowing software developers to create functions in a traditional programming language such as Java, which can then be invoked from either the condition or the action of a rule. Such functions may incorporate domain-specific (but reusable) logic.

Participants

There are generally three individuals having an interaction in an expert system. Primary among these is the end-user, the individual who uses the system for its problem solving assistance. In the construction and maintenance of the system there are two other roles: the problem domain expert who builds the system and supplies the knowledge base, and a knowledge engineer who assists the experts in determining the representation of their knowledge, enters this knowledge into an explanation module and who defines the inference technique required to solve the problem. Usually the knowledge engineer will represent the problem solving activity in the form of rules. When these rules are created from domain expertise, the knowledge base stores the rules of the expert system.

Inference rule

An inference rule is a conditional statement with two parts: an *if* clause and a *then* clause. This rule is what gives expert systems the ability to find solutions to diagnostic and prescriptive problems. An example of an inference rule is:

> If the restaurant choice includes French and the occasion is romantic,
> Then the restaurant choice is definitely Paul Bocuse.

An expert system's rulebase is made up of many such inference rules. They are entered as separate rules and it is the inference engine that uses them together to draw conclusions. Because each rule is a unit, rules may be deleted or added without affecting other rules - though it should affect which conclusions are reached. One advantage of inference rules over traditional programming is that inference rules use reasoning which more closely resembles human reasoning.

Thus, when a conclusion is drawn, it is possible to understand how this conclusion was reached. Furthermore, because the expert system uses knowledge in a form similar to the that of the expert, it may be easier to retrieve this information directly from the expert.

Procedure node interface

The function of the procedure node interface is to receive information from the procedures coordinator and create the appropriate procedure call. The ability to call a procedure and receive information from that procedure can be viewed as simply a generalization of input from the external world. In some earlier expert systems external information could only be obtained in a predetermined manner, which only allowed certain information to be acquired. Through the knowledge base, this expert system disclosed in the cross-referenced application can invoke any procedure allowed on its host system. This makes the expert system useful in a much wider class of knowledge domains than if it had no external access or only limited external access.

In the area of machine diagnostics using expert systems, particularly self-diagnostic applications, it is not possible to conclude the current state of "health" of a machine without some information. The best source of information is the machine itself, for it contains much detailed information that could not reasonably be provided by the operator.

The knowledge that is represented in the system appears in the rulebase. In the rulebase described in the cross-referenced applications, there are basically four different types of objects, with the associated information:

1. Classes: Questions asked to the user.

2. Parameters: Place holders for character strings which may be variables that can be inserted into a class question at the point in the question where the parameter is positioned.
3. Procedures: Definitions of calls to external procedures.
4. Rule nodes: Inferences in the system are made by a tree structure which indicates the rules or logic mimicking human reasoning. The nodes of these trees are called **rule nodes**. There are several different types of rule nodes.

The rulebase echoes a forest of many trees. The top node of the tree is called the *goal node*, in that it contains the conclusion. Each tree in the forest has a different goal node. The leaves of the tree are also referred to as rule nodes, or one of the types of rule nodes. A leaf may be an evidence node, an external node, or a reference node.

An *evidence node* functions to obtain information from the operator by asking a specific question. In responding to a question presented by an evidence node, the operator is generally instructed to answer "yes" or "no" represented by the numeric values 1 and 0 or provide a value between 0 and 1, representing a "maybe". Questions which require a response from the operator other than yes or no or a value between 0 and 1 are handled in a different manner.

A leaf that is an external node indicates that the data which will be used was obtained from a procedure call.

A *reference node* functions to refer to another tree or subtree.

A tree may also contain intermediate or minor nodes between the goal node and the leaf node. An intermediate node can represent logical operations like "And" or "Or".

The inference logic has two functions. It selects a tree to trace and then it traces that tree. Once a tree has been selected, that tree is traced, depth-first, left to right.

The word "tracing" refers to the action the system takes as it traverses the tree, asking classes (questions), calling procedures, and calculating confidences as it proceeds.

As explained in the cross-referenced applications, the selection of a tree depends on the ordering of the trees. The original ordering of the trees is the order in which they appear in the rulebase. This order can be changed, however, by assigning an evidence node an attribute "initial" which is described in detail in these applications. The first action taken is to obtain values for all evidence nodes which have been assigned an "initial" attribute. Using only the answers to these initial evidences, the rules are ordered so that the most likely to succeed is evaluated first. The trees can be further re-ordered because they are constantly being updated as a selected tree is being traced.

The type of information solicited by the system from the user by means of questions or classes should be tailored to the level of knowledge of the user. In many applications, the group of prospective uses is well-defined and the knowledge level can be estimated so that the questions can be presented at a level which corresponds generally to the average user. However, in other applications, knowledge of the specific domain of the expert system might vary considerably among the group of prospective users.

One application where this is particularly true involves the use of an expert system, operating in a self-diagnostic mode on a personal computer to assist the operator of the personal computer to diagnose the cause of a fault or error in either the hardware or software. In general, asking the operator for information is the most straightforward way for the expert system to gather information, assuming that the information is or should be within the operator's understanding. For example, in diagnosing a personal computer, the expert system must know the major functional components of the system. It could ask the operator, for instance, if the display is a monochrome or color display. The operator should, in all probability, be able to provide the correct answer. The expert system could, on the other hand, cause a test unit to be run to determine the type of display. The accuracy of the data collected by either approach in this instance probably would not be that different so the knowledge engineer could employ either approach without affecting the accuracy of the diagnosis. However, in many instances, because of the nature of the information being solicited, it is better to obtain the information from the system rather than asking the operator, because the accuracy of the data supplied by the operator is so low that the system could not effectively process it to a meaningful conclusion.

In many situations the information is already in the system, in a form that permits the correct answer to a question to be obtained through a process of inductive or deductive reasoning. The data previously collected by the system could include answers provided by the user to less complex questions previously asked for a different reason or results returned from test units that were previously run.

Real-time expert systems

Industrial processes, data networks, and many other systems change their state and even their structure over time. Real time expert systems are designed to reason over time and change conclusions as the monitored system changes. Most of these systems must respond to constantly changing input data, arriving automatically from other systems such as process control systems or network management systems.

Representation includes features for defining changes in belief of data or conclusions over time. This is necessary because data becomes stale. Approaches to this can include decaying belief functions, or the simpler validity interval that simply lets data and conclusions expire after specified time period, falling to "unknown" until refreshed. An often-cited example (attributed to real time expert system pioneer Robert L. Moore) is a hypothetical expert system that might be used to drive a car. Based on video input, there might be an intermediate conclusion that a stop light is green and a final conclusion that it is OK to drive through the intersection. But that data and the subsequent conclusions have a very limited lifetime. You would not want to be a passenger in a car driven based on data and conclusions that were, say, an hour old.

The inference engine must track the times of each data input and each conclusion, and propagate new information as it arrives. It must ensure that all conclusions are still current. Facilities for periodically scanning data, acquiring data on demand, and filtering noise, become essential parts of the overall system. Facilities to reason within a fixed deadline are important in many of these applications.

An overview of requirements for a real-time expert system shell is given in.[8] Examples of real time expert system applications are given in [9] and.[10] Several conferences were dedicated to real time expert system applications in the chemical process industries, including.[11]

Application

Expert systems are designed to facilitate tasks in the fields of accounting, medicine, process control, financial service, production, human resources, among others. Typically, the problem area is complex enough that a more simple traditional algorithm cannot provide a proper solution. The foundation of a successful expert system depends on a series of technical procedures and development that may be designed by technicians and related experts. As such, expert systems do not typically provide a definitive answer, but provide probabilistic recommendations.

An example of the application of expert systems in the financial field is expert systems for mortgages. Loan departments are interested in expert systems for mortgages because of the growing cost of labour, which makes the handling and acceptance of relatively small loans less profitable. They also see a possibility for standardised, efficient handling of mortgage loan by applying expert systems, appreciating that for the acceptance of mortgages there are hard and fast rules which do not always exist with other types of loans. Another common application in the financial area for expert systems are in trading recommendations in various marketplaces. These markets involve numerous variables and human emotions which may be impossible to deterministically characterize, thus expert systems based on the rules of thumb from experts and simulation data are used. Expert system of this type can range from ones providing regional retail recommendations, like Wishabi, to ones used to assist monetary decisions by financial institutions and governments.

While expert systems have distinguished themselves in AI research in finding practical application, their application has been limited. Expert systems are notoriously narrow in their domain of knowledge — as an amusing example, a researcher used the "skin disease" expert system to diagnose his rustbucket car as likely to have developed measles — and the systems are thus prone to making errors that humans would easily spot. Additionally, once some of the mystique had worn off, most programmers realized that simple expert systems were essentially just slightly more elaborate versions of the decision logic they had already been using. Therefore, some of the techniques of expert systems can now be found in most complex programs without drawing much recognition.

An example and a good demonstration of the limitations of an expert system is the Windows operating system troubleshooting software located in the "help" section in the taskbar menu. Obtaining technical operating system support is often difficult for individuals not closely involved with the development of the operating system. Microsoft has designed their expert system to provide solutions, advice, and suggestions to common errors encountered while using their operating systems.

Another 1970s and 1980s application of expert systems, which we today would simply call AI, was in computer games. For example, the computer baseball games Earl Weaver Baseball and Tony La Russa Baseball each had highly detailed simulations of the game strategies of those two baseball managers. When a human played the game against the computer, the computer queried the Earl Weaver or Tony La Russa Expert System for a decision on what strategy to follow. Even those choices where some randomness was part of the natural system (such as when to throw a surprise pitch-out to try to trick a runner trying to steal a base) were decided based on probabilities supplied by Weaver or La Russa. Today we would simply say that "the game's AI provided the opposing manager's strategy."

Advantages and disadvantages

- Compared to traditional programming techniques, expert-system approaches provide the added flexibility (and hence easier modifiability) with the ability to model rules as data rather than as code. In situations where an organization's IT department is overwhelmed by a software-development backlog, rule-engines, by facilitating turnaround, provide a means that can allow organizations to adapt more readily to changing needs.

- In practice, modern expert-system technology is employed as an adjunct to traditional programming techniques, and this hybrid approach allows the combination of the strengths of both approaches. Thus, rule engines allow control through programs (and user interfaces) written in a traditional language, and also incorporate necessary functionality such as inter-operability with existing database technology.

Disadvantages

- The Garbage In, Garbage Out (GIGO) phenomenon: A system that uses expert-system technology provides no guarantee about the quality of the rules on which it operates. All self-designated "experts" are not necessarily so, and one notable challenge in expert system design is in getting a system to recognize the limits to its knowledge.
- An expert system or rule-based approach is not optimal for all problems, and considerable knowledge is required so as to not misapply the systems.
- Ease of rule creation and rule modification can be double-edged. A system can be sabotaged by a non-knowledgeable user who can easily add worthless rules or rules that conflict with existing ones. Reasons for the failure of many systems include the absence of (or neglect to employ diligently) facilities for system audit, detection of possible conflict, and rule lifecycle management (e.g. version control, or thorough testing before deployment). The problems to be addressed here are as much technological as organizational.

Types of problems solved

Expert systems are most valuable to organizations that have a high-level of know-how experience and expertise that cannot be easily transferred to other members. They are designed to carry the intelligence and information found in the intellect of experts and provide this knowledge to other members of the organization for problem-solving purposes.

Typically, the problems to be solved are of the sort that would normally be tackled by a professional, such as a medical professional in the case of clinical decision support systems. Real experts in the problem domain (which will typically be very narrow, for instance "diagnosing skin conditions in teenagers") are asked to provide "rules of thumb" on how they evaluate the problem — either explicitly with the aid of experienced systems developers, or sometimes implicitly, by getting such experts to evaluate test cases and using computer programs to examine the test data and derive rules from that (in a strictly limited manner). Generally, expert systems are used for problems for which there is no single "correct" solution which can be encoded in a conventional algorithm — one would not write an expert system to find the shortest paths through graphs, or to sort data, as there are simpler ways to do these tasks.

Simple systems use simple true/false logic to evaluate data. More sophisticated systems are capable of performing at least some evaluation, taking into account real-world uncertainties, using such methods as fuzzy logic. Such sophistication is difficult to develop and still highly imperfect.

Shells or Inference Engine

A shell is a complete development environment for building and maintaining knowledge-based applications. It provides a step-by-step methodology, and ideally a user-friendly interface such as a graphical interface, for a knowledge engineer that allows the domain experts themselves to be directly involved in structuring and encoding the knowledge. Examples of shells include Drools, CLIPS, JESS, d3web, G2, eGanges, and OpenKBM (initially developed as a replacement for G2).

See also

- AI productions
- Artificial neural network
- Action selection mechanism
- Business Intelligence
- Business rules engine
- Case-based reasoning
- Connectionist expert system
- Decision support system
- Data Mining
- Fuzzy cognitive map
- Heuristic (computer science)
- Knowledge Acquisition and Documentation Structuring
- Knowledge base
- Knowledge engineering
- Machine learning
- OPS5
- Production system
- Rete algorithm
- Ripple down rules
- Self service software
- Type-2 fuzzy sets and systems

References

1. ^ ACM 1998, I.2.1
2. ^ Russell & Norvig 2003, pp. 22−24
3. ^ Luger & Stubblefield 2004, pp. 227–331
4. ^ Nilsson 1998, chpt. 17.4
5. ^ McCorduck 2004, pp. 327–335, 434–435
6. ^ Crevier 1993, pp. 145–62, 197−203
7. ^ *a b c d* One or more of the preceding sentences incorporates text from a publication now in the public domain: Ashford 1988
8. ^ "Process Control Using a Real Time Expert System", R. Moore, H. Rosenof, and G. Stanley, Proc. International Federation of Automatic Control (IFAC), Estonia, USSR, 1990 (http://www.gregstanleyandassociates.com/whitepapers/IFAC_Estonia_1990_Paper_Reforma
9. ^ "Experiences Using Knowledge-Based Reasoning in On-line Control Systems", G.M. Stanley, Proc. International Federation of Automatic Control (IFAC) Symposium on Computer-Aided Design in Control Systems, Swansea, UK, July, 1991 (http://www.gregstanleyandassociates.com/whitepapers/IFAC91objectPaper.pdf)

10. ^ "Real World Model-based Fault Management", R. Kapadia, G. Stanley, and M. Walker, Proceedings of the 18th International Workshop on the Principles of Diagnosis (DX-07), Nashville, TN, USA, June, 2007 (http://www.gregstanleyandassociates.com/dx07-final-submission.pdf)
11. ^ Proc. International Federation of Automatic Control (IFAC) Symposium on On-line Fault Detection and Supervision in the Chemical Process Industries, Newark, Delaware, April, 1992

Bibliography

- Ignizio, James (1991). *Introduction to Expert Systems*. ISBN 0-07-909785-5.
- Giarratano, Joseph C. and Riley, Gary (2005). *Expert Systems, Principles and Programming*. ISBN 0-534-38447-1.
- Jackson, Peter (1998). *Introduction to Expert Systems*. ISBN 0-201-87686-8.
- Walker, Adrian et al. (1990). *Knowledge Systems and Prolog*. Addison-Wesley. ISBN 0-201-52424-4.
- Darlington, Keith (2000). *The Essence of Expert Systems*. Pearson Education. ISBN 0-13-022774-9.
- US patent 4763277 (http://v3.espacenet.com/textdoc?DB=EPODOC&IDX=US4763277) , Ashford, Thomas J. *et al.*, "Method for obtaining information in an expert system", published 1988-08-09, issued 1988-08-09

External links

- Artificial Intelligence (http://www.dmoz.org/Computers/Artificial_Intelligence//) at the Open Directory Project

Retrieved from "http://en.wikipedia.org/wiki/Expert_system"
Categories: Artificial intelligence | Decision theory | Expert systems | Information systems