Expert Systems: An Introduction

K S R Anjaneyulu

K S R Anjaneyulu is a Research Scientist in the Knowledge Based Computer Systems Group at NCST. He is one of the authors of a book on rule based expert systems.

Expert systems encode human expertise in limited domains by representing it using *if-then* rules. This article explains the development and applications of expert systems.

Introduction

Imagine a piece of software that runs on your PC which provides the same sort of interaction and advice as a career counsellor helping you decide what education field to go into and perhaps what course to pursue. Or a piece of software which asks you questions about your defective TV and provides a diagnosis about what is wrong with it. Such software, called *expert systems*, actually exist. Expert systems are a part of the larger area of Artificial Intelligence.

One of the goals of Artificial Intelligence is to develop systems which exhibit 'intelligent' human-like behaviour. Initial attempts in the field (in the 1960s) like the General Problem Solver created by Allen Newell and Herbert Simon from Carnegie Mellon University were to create general purpose intelligent systems which could handle tasks in a variety of domains. However, researchers soon realized that developing such general purpose systems was too difficult and it was better to focus on systems for limited domains. Edward Feigenbaum from Stanford University then proposed the notion of expert systems. Expert systems are systems which encode human expertise in limited domains. One way of representing this human knowledge is using *If-then* rules. We will use this representation here to illustrate the basic ideas underlying expert systems.

Expert Systems are systems which encode human expertise in limited domains.

Components of an Expert System

A typical expert system consists of five components (*Figure 1*).

- the user interface

$$\begin{tikzcd}
\text{User Interface} \arrow[d, leftrightarrow] & & \\
\text{Explanation System} \arrow[d, leftrightarrow] & & \\
\text{Knowledge Base} \arrow[r] & \text{Inference Engine} \arrow[r, leftrightarrow] & \text{Working Memory}
\end{tikzcd}$$

Figure 1. Components of an expert system.

- the working memory
- the knowledge base
- the inference engine
- and the explanation system

The knowledge base and the working memory (WM) are the data structures which the system uses and the inference engine is the basic program which is used. The explanation system answers questions the user has and provides an explanation of its reasoning. Each of these components are briefly described below.

Working Memory

The working memory represents the set of facts known about the domain. The elements of the WM reflect the current state of the world. In an expert system, the WM typically contains information about the particular instance of the problem being addressed. For example, in a TV troubleshooting expert system, the WM could contain the details of the particular TV being looked at.

The actual data represented in the WM depends on the type of application. The initial WM, for instance, can contain a priori information known to the system. The inference engine uses this information in conjunction with the rules in the knowledge base to derive additional information about the problem being solved.

The working memory represents the set of facts known about the domain.

The knowledge base is a set of rules which represents the knowledge about the domain.

Knowledge Base

The knowledge base (also called rule base when *If-then* rules are used) is a set of rules which represents the knowledge about the domain. The general form of a rule is:

If cond1 **and** cond2 **and** cond3 ...
then action1, action2, ...

The conditions cond1, cond2, cond3, etc., (also known as antecedents) are evaluated based on what is currently known about the problem being solved (i.e., the contents of the working memory).

Each antecedent of a rule typically checks if the particular problem instance satisfies some condition. For example, an antecedent in a rule in a TV troubleshooting expert system could be: *the picture on the TV display flickers.*

The consequents of a rule typically alter the WM, to incorporate the information obtained by application of the rule. This could mean adding more elements to the WM, modifying an existing WM element or even deleting WM elements. They could also include actions such as reading input from a user, printing messages, accessing files, etc. When the consequents of a rule are executed, the rule is said to have been *fired*.

In this article we will consider rules with only one consequent and one or more antecedents which are combined with the operator **and**. We will use a representation of the form:

When the consequents of a rule are executed, the rule is said to have been *fired*.

rule id: **If** antecedent1 **and** antecedent2 .... **then** consequent

For instance, to represent the knowledge that if a person has a runny nose, a high temperature and bloodshot eyes, then one has a flu, we could have the following rule:

r1: **If** is(nose, runny) and is(temperature, high) **and** is(eyes, bloodshot)

then disease is flu

The inference engine is the program part of an expert system.

This representation, though simple, is often sufficient.

The disjunction (ORing) of a set of antecedents can be achieved by having different rules with the same consequent. Similarly, if multiple consequents follow from the conjunction (ANDing) of a set of antecedents, this knowledge can be expressed in the form of a set of rules with one consequent each. Each rule in this set will have the same set of antecedents.

Sometimes the knowledge which is expressed in the form of rules is not known with certainty (for example our **flu** rule is not absolutely certain). In such cases, typically, a degree of certainty is attached to the rules. These degrees of certainty are called *certainty factors*. We will not discuss certainty factors further in this article.

Inference Engine

The inference engine is the program part of an expert system. It represents a problem solving model which uses the rules in the knowledge base and the situation-specific knowledge in the WM to solve a problem.

Edward Feigenbaum – Father of Expert Systems

Given the contents of the WM, the inference engine determines the set of rules which should be considered. These are the rules for which the consequents match the current goal of the system. The set of rules which can be fired is called the *conflict set*. Out of the rules in the conflict set, the inference engine selects one rule based on some predefined criteria. This process is called *conflict resolution*. For example, a simple conflict resolution criterion could be to select the first rule in the conflict set.

A rule can be *fired* if all its antecedents are satisfied. If the value of an antecedent is not known (in the WM memory), the system checks if there are any other rules with that as a consequent; thus setting up a sub-goal. If there are no rules for that antecedent, the user is prompted for the value and the value is added to the WM. If a new sub-goal has been set up, a new set of rules will be considered in the next cycle. This process is repeated till, in a given cycle, there are no sub-goals or alternatively, the goal of the problem-solving has been derived.

This inferencing strategy is called *backward chaining* (since it reasons backward from the goal to be derived). There is another strategy, called *forward chaining* where the system works forward from the information it has in the working memory. In forward chaining, the conflict set will be created by rules which have all their antecedents true in a given cycle. The system continues till the conflict set becomes empty.

Explanation System

Expert systems typically need to be able to provide explanations regarding the conclusions they make. Most expert systems provide a mechanism whereby the user can ask questions about:

- why a particular question is being asked
- how the system came to a particular conclusion

Providing explanations is essential for the user to understand how the system works and determine whether its reasoning is correct or not.

Providing explanations is essential in all non-trivial domains for the user to understand how the system works and determine whether its reasoning is correct or not. Typically the system will keep track of what rules (knowledge) it is using and provide explanations based on a translation of these rules into English.

An Example

To illustrate the concepts we have been discussing so far, we will consider a simple decision making task – determining how to

r1: **If** gt(distance, 5)
then means is "drive"

r2: **If** gt(distance, 1) **and** lt(time, 15)
then means is "drive"

r3: **If** gt(distance, 1) **and** ge(time, 15)
then means is "walk"

r4: **If** is(means, "drive") **and** is(location, "city_centre")
then action is "take a taxi"

r5: **If** is(means, "drive") **and** is not(location, "city_centre")
then action is "drive your car"

r6: **If** is(means, "walk") **and** is(weather, "bad")
then action is "take an umbrella and walk"

r7: **If** is(means, "walk") **and** is(weather, "good")
then action is "walk".

gt - stands for greater than, ge - stands for greater than or equal to and lt - stands for less than

Figure 2. An example of a set of rules.

reach a particular location. The knowledge for this task is represented in the form of 7 rules given in *Figure 2*. The attributes used in these rules are: *means* (means of reaching destination), *distance* (distance of destination), *location* (location of destination), *time* (time available for travel) and *weather* (whether it is good or bad). The system is supposed to use the rules, when necessary take inputs from a user and recommend one of the following four actions to the user on how to reach a destination:

- take a taxi
- drive your car
- take an umbrella and walk
- walk

Box 1. Applications of Expert Systems

Expert systems can be created almost for any domain for which there exists a human expert. However, the domain should ideally be one in which an expert can tackle a task within a few hours. If it requires more time, it is likely that that the domain is too vast for current technology. Some of the expert systems which have been created are:

DENDRAL – Considered to be the first expert system. Identifies the molecular structure of unknown compounds. Developed by Stanford University.

MYCIN – A seminal expert system which made significant contributions to the field; but was not used in actual practice. Provides assistance to physicians in the diagnosis and treatment of meningitis and bacterial infections. Developed by Stanford University.

PROSPECTOR – Used successfully to locate deposits of several minerals, including copper and uranium. Developed by SRI International.

ALTREX – helps diagnose engine troubles of certain models of Toyota cars. Used in a central service department which can be called up by those actually servicing the cars for assistance, if required. Developed by their research lab.

PREDICTE – Given information about a high-rise building to be constructed, it provides estimates of the time required to construct it. Developed by Digital Equipment Corporation for use by Land Lease, Australia.

We simulate below the method which the system uses:

- The system's initial goal is to derive the value of the goal attribute *action*. So the conflict set initially contains all rules which have action as consequent. These rules are: r4, r5, r6 and r7. If the conflict resolution strategy uses the textual order of rules, r4 will be tried first.

- The value of action from r4 is 'Take a taxi' if the antecedent *means* is 'driving'. Since the value of *means* is not in the working memory (in fact the working memory is currently empty), it is made the new goal. Rules r1, r2, and r3 have consequent *means*. Based on our conflict resolution strategy, r1 needs to be tried first.

❖ r1 tests the magnitude of *distance*. Since the value of *distance* is not in the working memory and there is no rule which concludes it, the value for *distance* is obtained by asking the user and is added to the WM. Assume that the user gives the value 2 kms. 2 kms is less than 5 kms, and so r1 fails.

❖ The next rule with consequent *means*, r2, is now tried. First antecedent in r2 depends on *distance* and is true (since the working memory contains the value 2 for distance). Second antecedent in r2 depends on *time*. Again since *time* is not in the WM and there is no rule which deduces it, its value needs to be obtained from the user. Assume the user gives the value 5 minutes. This is added to the WM and based on this value the second antecedent succeeds. r2 concludes that *means* is driving.

❖ r3 fails because of the second antecedent.

❖ No other rules conclude about the value of *means*. The sub-goal to compute a value for *means* which was initiated by rule r4 is now complete. The inference engine focusses on r4 again.

❖ Based on the value of *means* in the WM, the first antecedent in r4 succeeds. The second antecedent is now checked. This involves location. Since there is no rule with consequent *location* and its value is not in the WM, the user is asked for the value. Let's assume the user gives the value as 'city centre'. This value is added to WM and r4 succeeds. A value for *action* is now obtained and is added into working memory.

❖ r5, r6 and r7 are tested, but each fails (try to see why). After r7 fails, the consultation is over. *Action* is marked as the goal attribute to be displayed and it has the value 'Take a taxi'. So the advice 'Take a taxi' is displayed.

This simple example illustrates the backward chaining strategy. A few observations are in order:

❖ The same strategy can be used even if additional rules are

Box 2. Fuzzy Expert Systems

To discuss fuzzy expert systems let us go back to the example we covered in the article. In real life, one may not always be able to give precise answers to questions such as:

How far is the destination?

How soon do you need to get to the destination?

For the first question, one may give the answer 'nearby', 'not too far', 'far' etc. For the second question, one may give the answer 'very quickly', 'quickly' etc.

How would expert systems cope with this? There are a class of expert systems called fuzzy expert systems which do fuzzy reasoning using the notion of fuzzy sets. One fundamental idea they use, is the notion of a membership set. Consider for example a fuzzy set for 'far'.

Far	Degree of Membership
0 kms	0.0
1 kms	0.1
2 kms	0.3
3 kms	0.6
4 kms	0.8
5 kms	0.9
...	...
10 kms	0.99

We map the distance to the destination to a degree of membership in the set 'far'. So instead of classifying the distance as either 'far' or 'not far', we have a continuous distribution of values. For instance, 0 kms which is not far has a degree of membership 0.0 and 10 kms which is definitely far has a degree of membership of say 0.99. Other distances have intermediate membership values. Using this, one could represent 'very near', 'near' etc, as having different degrees of membership in the set for 'far'.

Consequently the structure of the rules and the way the inference engine handles the rules would need to be changed. For instance, there are certain techniques which are used to combine the fuzziness from different antecedents.

added to the rule base. For instance, one could add another rule r8, which says that one should take a taxi if *means* is drive and *weather* is bad. This could be done without having to modify the other rules in the system. This illustrates the modularity of the knowledge in the system.

❖ The system strictly speaking is not generating advices. There are a set of four possible advices and it chooses one based on its reasoning. However, in this process, it adapts its interaction based on the rule it is trying to apply; thus exhibiting a goal-driven intelligent behaviour.

❖ The rules for this decision making task can be removed and substituted by rules for another task such as TV troubleshooting. Then using the same reasoning strategy one could create a TV troubleshooting system. So the strategy which we have outlined is domain independent. Here lies the strength of the expert systems approach.

Knowledge engineering or acquisition is the process of extracting knowledge about the domain in which the expert system is being created.

Knowledge Engineering

Knowledge engineering or acquisition is the process of extracting knowledge about the domain in which the expert system is being created. Typically this knowledge is obtained from a human expert in the domain. This knowledge is normally in the form of heuristic knowledge (rules of thumb) which the expert gains through experience over a period of time.

Knowledge engineering is the biggest bottleneck in the development of expert systems. How does one obtain this knowledge from an expert? It is difficult for an expert to explicitly state the knowledge he is using. For example, an experienced doctor would be able to diagnose general problems like jaundice, malaria etc. quite easily. However, if the doctor had to put that knowledge in the form of *If-then* rules, this would be much more difficult. During knowledge engineering, the doctor would be interviewed and posed representative cases. Based on his responses, the knowledge he is applying needs to be understood and encoded in the form of the knowledge representation used. The expert would then need to examine the behaviour of the system to see if the knowledge has been encoded properly. This

Knowledge engineering is the biggest bottleneck in the development of expert systems.

Box 3. Some Expert System Shells *

BABYLON is a development environment for expert systems. It includes frames, constraints, a prolog-like logic formalism, and a description language for diagnostic applications. It is implemented in Common Lisp and has been ported to a wide range of hardware platforms. Available by anonymous ftp from: ftp.gmd.de:/gmd/ai-research/Software/Babylon/ [129.26.8.84]

as a BinHexed stuffit archive, on the Web via the URL

http://www.gmd.de/

MIKE (Micro Interpreter for Knowledge Engineering) is a full-featured, free, and portable software environment designed for teaching purposes at the UK's Open University. It includes forward and backward chaining rules with user-definable conflict resolution strategies, and a frame representation language with inheritance and 'demons' (code triggered by frame access or change), plus user-settable inheritance strategies.

They are available by anonymous ftp from hcrl.open.ac.uk [137.108.81.16] as the files

MIKEv2.03: /pub/software/src/MIKEv2.03/*
MIKEv2.50: /pub/software/pc/MIKEV25.ZIP

ES: The October/November 1990 issue of BYTE also described the ES expert system. ES supports backward/forward chaining, fuzzy set relations, and explanation, and is a stand alone executable for IBM-PCs. ES is available by anonymous ftp from

ftp.uu.net:/pub/ai/expert-sys/ [192.48.96.9] as summers.tar.Z.

RT-Expert is a shareware expert system that lets C programmers integrate expert systems rules into their C or C++ applications. RT-Expert consists of a rule-compiler that compiles rules into C code, and a library containing the rule execution engine. RT-Expert for DOS works with Borland Turbo C, Borland C++, and Microsoft C/C++ compilers. The personal edition is licensed for educational, research, and hobby use. Applications created with RT-Expert personal edition are not licensed for commercial purposes. Professional editions are available for commercial applications using DOS, Windows, and Unix environments. RT-Expert is available by anonymous ftp from:

world.std.com:/vendors/rtis/rtexpert

For more information, write to Real-Time Intelligent Systems Corporation: rtis@world.std.com.

Box 3 continued...

Vidwan is a commercial backward chaining expert system which supports a rule based representation. It provides an interactive editor to input rules and provides explanation facilities. It also supports uncertainty reasoning. For information on Vidwan write to: The Vidwan Coordinator, National Centre for Software Technology, Gulmohar Cross Road No.9, Juhu, Mumbai 400 049. Email:

vidwan @saathi.ncst.ernet.in

*The information on public domain shells has been extracted from the FAQ on Expert System Shells created by Mark Kantrowitz, School of Computer Science, Carnegie Mellon University.

cycle would need to be continuously repeated until the system performs satisfactorily.

Depending on the complexity of the domain, knowledge engineering could take anywhere from a few days to a few years. Expert system tools have been created which provide support in the creation of this knowledge and carry out checks on the completeness and correctness of the knowledge represented in the system.

Creation of Expert Systems

How does one create an expert system? The best way to do this is to use an expert system shell. An expert system shell can be viewed as an expert system minus the domain knowledge (the analogy would be the difference between a database tool and a database system). It allows knowledge of a domain to be encoded in a specific format and put into the system to create expert systems for different domains. The advantage of using a shell is that it avoids the need for computer programming and allows the developer to focus only on the domain knowledge. This enables even non-computer professionals to create expert systems. For instance, one could create a system for the example described in this article by just keying in the rules which were given. The shell would provide the interface, the inference engine and the explanation system.

An expert system shell can be viewed as an expert system minus the domain knowledge.

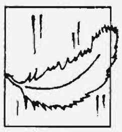
Conclusions

In this article, we have just scratched the surface of the wide area of expert systems which have turned out to be the most commercial aspect of Artificial Intelligence. A large number of expert systems are in real use and quite a few even being sold for individual use. In the future one is likely to see more expert systems packaged with domain knowledge being sold. Further, these systems are also likely to carry out specialized tasks as parts of much larger software systems.

Suggested Reading

- Virgil Negoita. *Expert Systems and Fuzzy Systems*. Benjamin Cummings, 1985.
- Edward Feigenbaum, Pamela McCorduck and H Penny Nii. *The Rise of the Expert Company*. Vintage Books, 1989.
- Elaine Rich and Kevin Knight. *Artificial Intelligence*. Tata McGraw-Hill, 1991.
- M Sasikumar, S Ramani, S Muthu Raman, KSR Anjaneyulu and R Chandrasekar. *Rule Based Expert Systems: A Practical Introduction*. Narosa Publishing House, 1993.

Address for Correspondence
K S R Anjaneyulu
Knowledge Based Computer Systems Group
National Centre for Software Technology
Mumbai 400 049, India.
anji@saathi.ncst.ernet.in



"Congratulations ... you're the first victim of recombinant DNA"

From: *Gene Antics*