EXPERT SYSTEMS

Peter Lucas
Institute for Computing and Information Sciences, Radboud University Nijmegen, the Netherlands

Keywords: Expert Systems, Knowledge-based Systems, Knowledge Systems, Knowledge Engineering, Knowledge Management, Horn Clauses, Objects, Frames, Rules, Bayesian Belief Networks, Influence Diagrams, Problem Solving, Inference Engine, Inference, Automated Reasoning, Abduction, Deduction, Consistency-based Reasoning, Inheritance

Contents

Summary

Expert systems, also called knowledge-based systems or knowledge systems, are computer systems characterized by the fact that an explicit distinction is made between a part in which knowledge of a problem domain is represented, and a part which manipulates that knowledge to reason about or solve an actual problem using problem data. Both the type of knowledge used in solving the problem and the nature of the problem-solving methods used determine which problems can be solved.

The knowledge represented in a knowledge base is formal in nature, and is the result of modeling essential features of the domain for the problem at hand. The incorporated knowledge may be acquired from domain experts, literature or datasets. Designing an expert system usually involves using methodologies for knowledge acquisition, modeling and evaluation.

1. Introduction

In this section, the field of expert systems is introduced and defined. Furthermore, two early examples of expert systems are discussed.

1.1 Definition of the Field

The phrase *knowledge-based system*, or *knowledge system*, is generally employed to denote information systems in which some symbolic representation of human knowledge of a domain is applied, usually in a way resembling human reasoning, to solve actual problems in the domain. Examples of problem domains include trouble shooting of equipment, medical diagnosis, financial advice, product design and so on. As this knowledge is often derived from experts in a particular field, and early knowledge-based systems were actually developed in close collaboration with experts, the term *expert system* was the term used in the early days to refer to these systems. Knowledge, however, can also be extracted from literature or from a datasets by using machine-learning methods. Moreover, not all domains of specific expert systems may be viewed as specialists' fields. As a consequence, some people prefer to make a distinction between expert systems and knowledge-based systems - in their view the latter are more general than the former as the former should always concern a specialist's field. In this chapter such a distinction will not be made as the techniques used in knowledge-based systems and the ones used in building expert systems are identical. Hence, the terms 'expert system' and 'knowledge-based system' will be used interchangingly.

Present generation expert systems are capable of dealing with restricted problem domains. Gathering, maintaining and updating the incorporated knowledge taking into account its associated context, such as working environment, organization and field of expertise belongs to an area referred to as *knowledge management*. The art of developing an expert system is called *knowledge engineering*, when there is emphasis on the pragmatic engineering aspects, or *knowledge modeling*, when development of domain models is emphasized. The latter is strictly speaking part of the former. The process of collecting and analyzing knowledge in a problem domain is called *knowledge acquisition*, or *knowledge elicitation* when the knowledge is gathered from interviews with experts, normally using interview techniques as developed by psychologists.

1.2 Origin and Evolution

One of the first systems with which the phrase expert system has been associated, is *Heuristic DENDRAL*. The DENDRAL project commenced in 1965 at Stanford University. The system was developed by J. Lederberg, an organic chemist (and Nobel prize winner in chemistry), in conjunction with E.A. Feigenbaum and B.G. Buchanan, both well-known research scientists in artificial intelligence at the time. The Heuristic DENDRAL system offered assistance in the field of organic chemistry in determining the structural formula of a chemical compound that has been isolated from a given sample. In determining a structural formula, information concerning the chemical formula, such as $C_4 H_9OH$ for butanol, and the source the compound has been taken

from, is used as well as information that has been obtained by subjecting the compound to physical, chemical and spectrometric tests.

The original DENDRAL algorithm was developed by J. Lederberg for generating all possible isomers of a chemical compound. Heuristic DENDRAL contains a subsystem, the so-called Structure Generator, which implements the DENDRAL algorithm, but in addition incorporates various heuristic constraints on possible structures, thus reducing the number of alternatives to be considered by the remainder of the system. Heuristic DENDRAL helps in interpreting the patterns in a spectrogram. It contains a considerable amount of chemical knowledge to do so. To this end, another subsystem of Heuristic DENDRAL, called the Predictor, suggests expected mass spectrograms for each molecular structure generated by the Structure Generator. Each expected mass spectrogram is then tested against the mass spectrogram observed using some measure of similarity for comparison; this has been implemented in the last part of the system, the Evaluation Function. Usually, more than one molecular structure matches the pattern found in the spectrogram. Therefore, the system usually produces more than one answer, ordered by the amount of evidence favoring them.

The best-known expert system of this early period of the field is *MYCIN*, a system developed by E.H. Shortliffe when at Stanford University at the end of the 1970s. The MYCIN system was capable of assisting physicians in the diagnosis and treatment of some infectious diseases, in particular meningitis and bacterial septicemia. When a patient shows the signs of such a disease, a culture of blood and urine is made in order to determine which bacterium species caused the infection. Usually, it takes 24 to 48 hours before the laboratory results are known. Often, however, the physician cannot wait that long before starting treatment, since otherwise the disease will progress and actually cause the death of the patient. MYCIN gives an interim indication of the organisms most likely responsible for the patient's disease on the basis of the (possibly incomplete and inexact) patient data available to the system. Given this indication, MYCIN advises on the administration of appropriate drugs to control the infection. Again, the system was only capable of doing so by drawing upon a considerable body of formalized expert knowledge in infectious disease. The MYCIN system clearly left its mark on the expert systems that have been developed since. Even today, this expert system and its derivatives are a source of inspiration for expert system researchers.

2. Expert System Principles

As an expert system is a software system, the structure of expert systems can be described and understood in terms of the components of which such systems consist, as well as in terms of the interchange of information between these components. This is called an *architecture*. These software issues are summarized in Section 2.1. Section 2.2. pays attention to expert systems as programs solving particular types of problem.

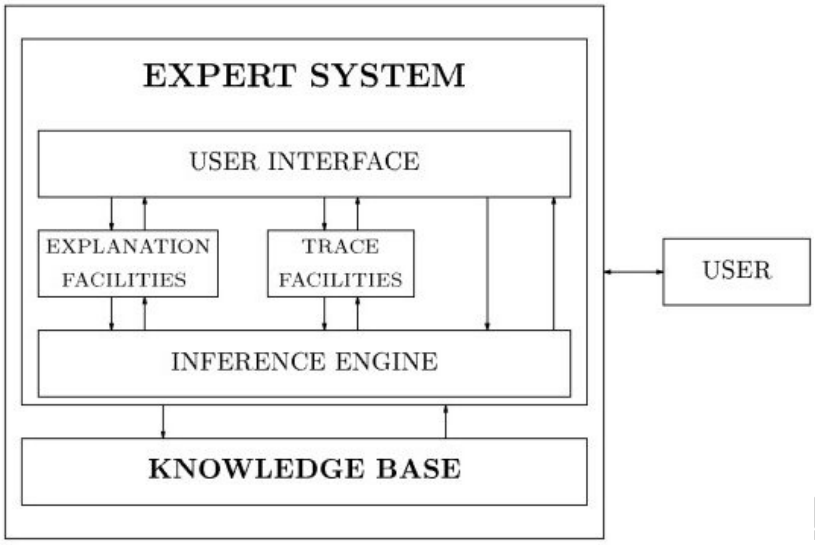

Figure 1: Global architecture of an expert system

2.1 Expert System Architecture

In the early years, expert systems were written in a high-level programming language, usually in LISP. However, using such a programming language as an expert system building tool, demands disproportionate attention to the implementational aspects of the system unrelated to the problem domain. Moreover, the expert knowledge of the domain and the algorithms for applying this knowledge will become densely interwoven. This leads to systems that, once constructed, are practically not adaptable to changing views about the domain of concern. Expert knowledge, however, is dynamic: knowledge and experience are continuously changing, requiring modifications of the corresponding expert system. Attempts to solve this problem led to the view that the domain knowledge and algorithms for applying this knowledge should be separated explicitly. This principle constitutes the paradigm of today's expert system design:

expert system = knowledge + problem-solving methods.

Accordingly today's expert systems typically have two basic components as shown in Figure 1:

- A *knowledge base* that captures the domain-specific knowledge, and
- An *inference engine* that consists of algorithms for manipulating the knowledge represented in the knowledge base to solve a problem presented to the system.

In addition, an expert system may contain facilities to explain, illustrate or offer documentation for its reasoning steps, often called *explanation facilities*. During the development of a system it might be worthwhile to trace the reasoning behavior in more detail, which is provided by a *trace facility*. The capabilities of an inference engine are typically used to implement particular problem-solving methods, for example methods to solve diagnostic problems.

Modern expert systems are rarely written in a high-level programming language. Instead, they are built in a special software environment, known under various names like *expert system shells*, *expert-system builder tools*, or *knowledge-based system toolkit*. An early example of such an environment is EMYCIN (Essential MYCIN), a system that emerged from MYCIN by stripping it of its knowledge concerning infectious disease. Other, more recent examples, include CLIPS, JESS, AION-DS. Also the PROLOG programming language is eminently suitable to implement expert systems.

Every expert system shell or builder tool offers a formal language, called a *knowledge-representation formalism*, for encoding the domain knowledge in the knowledge base. Furthermore, they provide one or more inference engines that are capable of manipulating knowledge that is represented in the formalism. The developer of an expert system is therefore shielded from most of the system's algorithmic aspects; only the domain-specific knowledge has to be provided and expressed in the knowledge-representation formalism, whereas the reasoning as offered by the tool may need to be tailored to the type of problem solving required. Note that several advantages arise from the property that a knowledge base can be developed separately from the inference engine. A knowledge base can be developed and refined stepwise. Errors and inadequacies can be easily remedied without making changes to the program text necessary. Furthermore, an expert-system builder tool can be used to develop expert systems for different problem domains, which may save in development time and costs.

2.2 Problem-solving Methods

As said above, the inference engine of an expert system shell is normally customized to obtain more specific problem-solving methods. An example is a diagnostic method that is able to use causal knowledge about the relationship between causes and the associated effects to explain observed malfunction of a device in terms of possible causes of that malfunction. Sometimes the same problem can be solved in different ways, using different types of knowledge and different methods. For example, the faults of a device can also be diagnosed by using expert knowledge saying that a particular combination of findings is typical for the occurrence of a particular fault. In this case, so-called *heuristic associations* rather than *causal knowledge* is used to diagnose malfunction. More about this will be said below where the formal properties of some methods will be examined. Typical examples of problems for which specific methods have been developed are:

- diagnosis
- prediction
- planning and scheduling
- design
- decision making

3. Knowledge Representation and Inference

Key issues for the suitability of any expert-system builder tool are the features it offers to model particular problem domains. In particular the knowledge-representation

formalism and the types of reasoning supported are of major importance. Logic, probability theory and decision theory are sufficiently general to permit describing the nature of knowledge representation, inference and problem solving without having to resort to special-purpose languages.

3.1 Logic Representation and Reasoning

Expert systems usually offer a number of different ways to represent knowledge in a domain, and to reason with this knowledge automatically to derive conclusions. Although the languages offered by actual systems and tools may differ in a number of ways, there are also many similarities. The aspects that the languages have in common can be best understood in terms of a logical representation, as accomplished below. The role and place of logic in artificial intelligence is discussed in more detail in *Artificial Intelligence: Definition, Trends, Techniques, and Cases*: Section 1.4, and in *Logic in AI*.

3.1.1. Horn-clause Logic

A *Horn clause* or *rule* is a logical implication of the following form

$$\forall x_1 \cdots \forall x_m \left(\left(A_1 \wedge \cdots \wedge A_n \right) \rightarrow B \right) \qquad (1)$$

where A_i, B are literals of the form $P\left(t_1, \ldots, t_q\right)$, i.e. without a negation sign, representing a relationship P between terms t_k, which may involve one or more universally quantified variables x_j, constants and terms involving function symbols. As all variables in rules are assumed to universally quantified, the universal quantifiers are often omitted if this does not give rise to confusion. If $n = 0$, then the clause consists only of a conclusion, which may be taken as a *fact*. If, on the other hand, the conclusion B is empty, indicated by $\perp$, the rule is also called a *query*. If the conditions of a query are satisfied, this will give rise to a contradiction or inconsistency, denoted by $\perp$, as the conclusion is empty. So, an empty clause means actually inconsistency.

A popular method to reason with clauses, and Horn clauses in particular, is *resolution*. Let $\mathcal{R}$ be a set of rules not containing queries, and let $Q \equiv \left(A_1 \wedge \cdots \wedge A_n \right) \rightarrow \perp$ be a query, then

$$\mathcal{R} \cup \{Q\} \vdash \perp$$

where $\vdash$ means the application of resolution, implies that the conditions

$$\forall x_1 \cdots \forall x_m \left(A_1 \wedge \cdots \wedge A_n \right)$$

are not all satisfied. Since resolution is a sound inference rule, meaning that it respects the logical meaning of clauses, it also holds that $\mathcal{R} \cup \{Q\} \models \perp$, or equivalently

$$\mathcal{R} \models \exists x_1 \cdots \exists x_m \left(A_1 \wedge \cdots \wedge A_n \right)$$

if $\mathcal{R}$ only consists of Horn clauses. This last interpretation explains why deriving inconsistency is normally not really the goal of using resolution; rather, the purpose is to derive certain facts. Since resolution is only complete for deriving inconsistency, called *refutation completeness*, it is only safe to 'derive' knowledge in this indirect manner. There exist other reasoning methods which do not have this limitation. However, resolution is a simple method that is understood in considerable depth. As a consequence, state-of-the-art resolution-based reasoners are very efficient. Resolution can also be used with clauses in general, which are logical expressions of the form

$$(A_1 \wedge \cdots \wedge A_n) \rightarrow (B_1 \vee \cdots \vee B_m)$$

usually represented as:

$$\neg A_1 \vee \cdots \vee \neg A_n \vee B_1 \vee \cdots \vee B_m$$

where all variables are again assumed to be universally quantified.
Rules of the form (1) are particularly popular as the reasoning with propositional Horn clauses is known to be possible in linear time, whereas reasoning with propositions or clauses in general (where the right-hand side consists of disjunctions of literals) is known to be NP complete, i.e. may require time exponential in the size of the clauses. Note that allowing negative literals at the left-hand site of a rule is equivalent to having disjunctions at the right-hand side. Using a logical language that is more expressive than Horn-clause logic is sometimes unavoidable, and special techniques have been introduced to deal with their additional power.

Let KB be a knowledge base consisting of a set (conjunction) of rules, and let F be a *set of facts* observed for a particular problem $\mathcal{P}$, then there are generally three ways in which a problem can be solved, yielding different types of solutions. Let $\mathcal{P}$ be a problem, then there are different classes of solutions to this problem:

- **Deductive solution:** S is a *deductive solution* of a problem $\mathcal{P}$ with associated set of observed findings F if

$$KB \cup F \models S \qquad (2)$$

and $KB \cup F \not\models \bot$, where S is a set of solution formulae.

- **Abductive/inductive solution:** S is an *abductive solution* of a problem $\mathcal{P}$ with associated set of observed findings F if the following *covering condition*

$$KB \cup S \cup K \models F \qquad (3)$$

is satisfied, where K stands for *contextual knowledge*. In addition, it must hold that $KB \cup S \cup C \not\models \bot$ (consistency condition), where C is a set of logical constraints on solutions. For the abductive case, it is assumed that the knowledge base KB contains a logical representation of *causal knowledge* and S consists of facts; for the inductive

case, KB consists of background facts and rules, and S, called an *inductive solution*, consists of rules.

- **Consistency-based solution**: S is a *consistency-based solution* of a problem $\mathcal{P}$ with associated set of observed findings F if

$$KB \cup S \cup F \not\models \bot \qquad (4)$$

Note that a deductive solution is a consistent conclusion that follows from a knowledge base KB and a set of facts, whereas an abductive solution acts as a hypothesis that *explains* observed facts in terms of causal knowledge, i.e. cause-effect relationships. An inductive solution also explains observed facts, but in terms of any other type of knowledge. A consistency-based solution is the weakest kind of solution, as it is neither required to be concluded nor is it required to explain observed findings.

-
-
-

Bibliography

Brachman R.J. (1983). What IS-A is and isn't: an analysis of taxonomic links in semantic networks. IEEE Computer, vol. 16, no. 10, pp. 30-36. [This is a classic paper about the meaning of the IS-A relation as used in taxonomies.]

Buchanan B.G., Shortliffe E.H. (1984). Rule-based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. Addison-Wesley, Reading, MA. [Collection of papers about the many different aspects of the MYCIN project.]

Buchanan BG, Sutherland G.L., Feigenbaum E.A. (1969). Heuristic DENDRAL: a program for generating explanatory hypotheses in organic chemistry. In: Machine Intelligence (B. Meltzer, D. Michie, eds.), 4, Edinburgh University Press, Edinburgh. [Description of the early expert system DENDRAL.]

Bylander T, Allemang D., Tanner M.C., Josephson J.R. (1992). The computational complexity of abduction. In *Knowledge Representation* (R.J. Brachman, H.J. Levesque and R. Reiter, eds.), pp. 25-60. The MIT Press, Cambridge, MA. [Ground-breaking paper about the computational complexity of different abductive methods.]

Clancey W.J. (1985). Heuristic classification. Artificial Intelligence, vol. 27, pp. 289-350. [Paper analyzing reasoning with heuristic association rules.]

Console L., Theseider Dupré D.,Torasso P. (1989). A theory of diagnosis for incomplete causal models. In: Proceedings of the 10th International Joint Conference on Artificial Intelligence, pp. 1311-1317. [Paper introducing the use of incompleteness literals in abductive diagnosis.]

Cooper G.F. (1988). A method for using belief networks as influence diagrams. In: Proceedings of the 4th Workshop on Uncertainty in Artificial Intelligence, pp. 55-63. [Paper proposing an algorithm to reason with an influence diagram, based on a Bayesian-belief-network algorithm.]

Forbus K.D., De Kleer J. (1993). Building Problem Solvers. The MIT Press, Cambridge, MA. [Book about the principles of and the building of problem solvers, in particular consistency-based ones.]

Glymour C., Cooper G.F. (1999). Computation, Causation & Discovery. MIT Press, Menlo Park, CA. [Collection of papers about discovering Bayesian belief networks and causal theories from data.]

De Kleer J., Williams B.C. (1987). Diagnosing multiple faults. Artificial Intelligence, vol. 32, pp. 97-130. [First paper describing the basic ideas behind consistency-based diagnosis.]

Lauritzen S.L., Spiegelhalter D.J. (1987). Local computations with probabilities on graphical structures and their application to expert systems. Journal of the Royal Statistical Society (Series B), vol. 50, pp. 157-224. [Paper introducing in 1987 a new method for probabilistic reasoning with Bayesian belief networks, now widely used.]

Lucas P.J.F., Van der Gaag L.C. (1991). Principles of Expert Systems. Addison-Wesley, Wokingham, UK. [Textbook on expert systems that emphasizes knowledge representation and inference, including formal meanings of knowledge-representation formalisms.]

Pearl J. (1988). Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, Palo Alto. [First, ground-breaking book about Bayesian belief networks.]

Peng Y, Reggia J.A. (1990). Abductive inference models for diagnostic problem solving. Springer-Verlag, New York. [Book describing a restrictive method for abductive diagnosis using set theory.]

Poole D. (1989). Explanation and prediction: an architecture for default and abductive reasoning. Computational Intelligence, vol. 5, no. 2, pp. 97-110. [Paper describing a logical approach to diagnostic reasoning.]

Reiter R. (1987). A theory of diagnosis from first principles. Artificial Intelligence, vol. 32, pp. 57-95. [First paper formalizing consistency-based diagnosis.]

Schreiber A.Th., Akkermans H., Anjewierden A., De Hoogh R., Shadbolt N., Van de Velde W., Wielinga B. (2000). Knowledge Engineering and Management: The CommonKADS Methodology. MIT Press, Menlo Park, CA. [Book presenting an overview of the CommonKADS knowledge-engineering methodology.]

Shachter R.D. (1986). Evaluating influence diagrams. Operation Research, vol. 34, no. 6, pp. 871-882. [First paper proposing an algorithm to manipulate influence diagrams for the purpose of decision making.]

Biographical Sketch

Dr. Peter J.F. Lucas is an associate professor with the Institute of Computing and Information Sciences at the Radboud University Nijmegen, the Netherlands. He has been both educated as a medical doctor and a computer scientist, and has been involved in research in the area of Artificial Intelligence, in particular knowledge-based systems, since the beginning of the 1980s. He has contributed to this area by theoretical as well as applied research, the latter for the major part focused on the field of medicine. At the moment, his research interests include topics such as: model-based diagnosis, Bayesian networks and decision theory, applied logic and theorem proving, and statistical machine learning. He is also still performing research in Biomedical Informatics.