# EXPERT SYSTEMS[1]

**Peter Lucas**, Institute for Computing and Information Sciences,
Radboud University, Nijmegen, The Netherlands,
E-mail: peterl@cs.ru.nl

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

## Contents

**Glossary** **2**

**Summary** **6**

**1 Introduction** **6**
1.1 Definition of the Field . . . . . 6
1.2 Origin and Evolution . . . . . 7

**2 Expert System Principles** **7**
2.1 Expert System Architecture . . . . . 8
2.2 Problem-solving Methods . . . . . 9

**3 Knowledge Representation and Inference** **9**
3.1 Logic Representation and Reasoning . . . . . 10
3.1.1 Horn-clause Logic . . . . . 10
3.1.2 Objects, Attributes and Values . . . . . 12
3.2 Diagnostic Problem Solving . . . . . 13
3.2.1 Deductive Diagnosis . . . . . 13
3.2.2 Abductive Diagnosis . . . . . 14
3.2.3 Consistency-based Diagnosis . . . . . 15
3.3 Configuring and Design . . . . . 17
3.4 Uncertainty . . . . . 19
3.5 Decision Making . . . . . 22

**4 Knowledge Engineering** **23**

**5 Conclusions** **25**

[1]Published in the *Encyclopedia of Life Support Systems* of UNESCO.

## Glossary

**Abduction:** Type of reasoning with causal knowledge in which observations are explained in terms of represented cause-effect relationships.

**Attribute:** Feature of an object or a thing; a description of an object or a thing is obtained by inspecting its attributes with its associated values, possibly extended by those obtained by inheritance.

**Bayesian belief network:** Representation of a joint probability distribution $\Pr(X_1, \ldots, X_n)$ defined on random variables $X_i$, for $i = 1, \ldots, n$, as nodes in a directed acyclic graph. The joint probability distribution is defined in terms of a set of (conditional) probability distributions $\Pr(X_i \mid \pi(X_i))$, where $\pi(X_i)$ corresponds to the set of parent nodes of the node corresponding to the variable $X_i$. Formally, it holds that:

$$\Pr(X_1, \ldots, X_n) = \prod_{i=1}^{n} \Pr(X_i \mid \pi(X_i))$$

By setting variables to particular values, representing in this way the effect of observing evidence $E$, an updated probability distribution can be computed. If $\Pr^E$ is the updated probability distribution, then $\Pr^E(V_i)$ gives the updated probability distribution, taking the effect of $E$ into account. It holds that $\Pr^E(V_i) = \Pr(V_i \mid E)$, i.e. the original probability distribution with respect to $V_i$, conditioned on the evidence $E$.

**Clause:** Logical formula of the form $L_1 \vee L_2 \vee \cdots \vee L_n$, where $L_i$ is either equal to an atomic formula $A_i$ or its negation $\neg A_i$. Atomic formulas may contain terms, including variables; the variables then have the meaning of universally quantified variables.

**Completeness:** Any formula $\varphi$ which follows logically from another formula $\psi$, will also be derived using logical (syntactical) inference rules; formally, if $\psi \vDash \varphi$ then $\psi \vdash \varphi$. Note that if a logic is sound (see there) and complete, one can use either the notation $\psi \vDash \varphi$ or $\psi \vdash \varphi$, as these are then equivalent, although the meanings of these two notions is still different.

**Conditional independence:** Two random variables $A$ and $B$ are called conditionally independent given a third random variable $C$ if it holds that $\Pr(A \mid B, C) = \Pr(A \mid C)$, or equivalently, if it holds that $\Pr(B \mid A, C) = \Pr(B \mid C)$, where Pr is a probability distribution. In words: any uncertain knowledge about $B$ given that we already know $C$ does not affect our knowledge about $A$, and any uncertain knowledge about $A$ given that we already know $C$ does not affect our knowledge about $B$. See also the item 'independence'.

**Consistency:** a logical formula $\varphi$ is called consistent when none of the formulas which logically follow from it contradict; formally: $\varphi \nvdash \bot$, or equally $\varphi \nvDash \bot$ if the logic is sound and complete (see there).

**Decision strategy:** Optimal sequence of decisions of actions, determined by taking both the uncertainty of observations and outcome of actions into account, in addition to costs of the observations, outcomes, decisions or actions.

**Decision theory:** Mathematical theory of the process of making rational choices under conditions of uncertainty.

**Deduction:** procedure consisting of the application of logical, syntactical inference rules to logical formulas, which, when they are applicable, yield new formulas. Formal notation: if $\mathcal{I}$ stands for the collection of inference rules used, then the deduction of formula $\varphi$ from $\psi$ is denoted by $\psi \vdash_{\mathcal{I}} \varphi$. This may involve several inference steps, and different inference rules. If the used collection of inference rules $\mathcal{I}$ is clear from the context, $\vdash_{\mathcal{I}}$ is usually written as $\vdash$.

**Expert system:** Information system that incorporates significant portions of human knowledge, often of experts in a particular domain, normally represented in a separate part of the system, called a knowledge base (see there). Also called knowledge-based system or knowledge system.

**Expert-system shell:** Computer program which when supplied with a particular knowledge base will yield an expert system. The minimum capability of an expert-system shell is reasoning or inference with the knowledge within the knowledge base, producing conclusions based on entered findings or facts.

**Expert-system builder tool:** Computer program that is used in building the knowledge base of an expert system.

**Explanation facility:** Component of an expert system that is used to explain the reasoning process that gave rise to certain conclusions or questions to the user.

**Frame:** Synonymous with object.

**Heuristic:** Particular trick, often based on deep knowledge of a domain, which helps in achieving particular goals, such as finding the solution of a particular problem, which otherwise would not have found at all or not in reasonable time.

**Horn clause:** Clause that contains at most one positive literal. Different name are linked to Horn clauses of different forms; Horn clauses of the form $A_1 \wedge \cdots \wedge A_n \rightarrow B$ are called *rules*, those of the form $A_1 \wedge \cdots \wedge A_n \rightarrow \bot$ are called *queries*, and those of the form $\rightarrow B$ are called *facts*.

**Inconsistency:** Deduction of a contradiction between formulas deducible from a formula $\psi$; formally: $\psi \vdash \bot$, or equally, for a sound and complete logic, $\psi \vDash \bot$

**Independence:** Two random variables $A$ and $B$ are called (unconditionally) independent if it holds that $\Pr(A \mid B) = \Pr(A)$ (in words: knowing anything about $B$ does not affect our knowledge about $A$), or equally, if $\Pr(B \mid A) = \Pr(B)$, where Pr is a probability distribution. See also 'conditional independence'.

**Inference:** The process of deriving knowledge from given knowledge. If this knowledge is represented in the form of standard logic, then this term is synonymous with deduction.

**Inference engine:** Inference algorithm, or computer implementation of it in a programming language, that allows carrying out inference steps to derive knowledge from knowledge automatically.

**Influence diagram:** Graph-based representation closely related to a Bayesian belief network, but which contains in addition to chance nodes, also decision nodes, modeling making decisions, and a utility node, modeling a utility function. Solving a influence diagram amounts to finding an optimal decision strategy, i.e. the utility function is being optimized, taking into account the probabilistic and utility information represented in an influence diagram.

**Inheritance:** An object $O$ with attributes $a_1, \ldots, a_n$ with values $c_1, \ldots, c_n$, which is known to be a subobject of (less general than) object $O'$, can be represented in first-order predicate logic as follows:

$$\forall x(O(x) \rightarrow (O'(x) \wedge a_1(x) = c_1 \wedge \cdots \wedge a_n(x) = c_n))$$

Similarly, if object $O'$ is known to be the most generic object in a domain with attributes $b_1, \ldots, b_m$, this can be represented as follows:

$$\forall x(O'(x) \rightarrow (b_1(x) = d_1 \wedge \cdots \wedge b_m(x) = d_n))$$

If these formulas are denoted by KB (knowledge base), and it is known that $i$ is an instance of $O$, then obviously:

$$(\mathrm{KB} \wedge O(i)) \vDash (O'(i) \wedge b_1(i) = d_1 \wedge \cdots \wedge b_m(i) = d_n \wedge a_1(i) = c_1 \wedge \cdots \wedge a_n(i) = c_n)$$

**Knowledge base:** Store of knowledge of a domain, specified in a knowledge-representation formalism, and used to handle specific problems in a problem domain, such diagnosing disease, predicting weather or developments in the financial markets, using an expert-system shell.

**Knowledge-based system:** Synonymous with expert system. Some people make a distinction between expert systems, which they see as systems that contain expert knowledge, whereas they see knowledge-based systems as systems that may contain any sort of knowledge. As there are no hard criteria to judge whether knowledge is expert knowledge or not, this distinction is not very useful, and is therefore not made here.

**Knowledge engineering:** Process of developing an expert system.

**Knowledge-representation formalism:** Formal language that can be used to represent knowledge. A typical example of a knowledge-representation formalism is predicate logic. Also programming languages can be seen as knowledge-representation languages, although may of those languages have not been designed to represent every-day human knowledge well, but rather as language which allow for representing specialized knowledge of how to control a machine.

**Marginal probability distribution:** If $\Pr(V_1, \ldots, V_n)$ is a joint probability distribution defined on the variables $V_i$, for $i = 1, \ldots, n$, then a probability distribution for any proper subset of these variables is called a marginal probability distribution. It holds (for discrete probability distributions) that:

$$\Pr(V_1, \ldots, V_{n-1}) = \sum_{V_n} \Pr(V_1, \ldots, V_n)$$

i.e. by summation over the domain of $V_n$. This marginalization rule can be applied recursively to compute any marginal probability distribution from the original joint probability distribution.

**Object:** In the context of expert systems, an object is usually understood to be a structured attribute description of a thing. Often special-purpose languages are used to represent objects, but from a semantic point of view, objects in many different languages for objects can be understood in terms of Horn-clause logic. This meaning is different from that in object-oriented programming, where an object is an active process that is normally able to communicate by exchanging messages with other objects. However, objects are specified using languages closely related to those used in expert systems. Hence, the distinction is not always clear, in particular when the capabilities of objects within expert systems have been extended into the direction of object-oriented programming languages.

**Ontology:** Description of the objects in domain, usually with the purpose of precisely describing the terminology in a domain. There is some relationship with a frame taxonomy, but ontologies are normally understood in a less formal way.

**Reasoning:** See inference.

**Resolution:** Special logical inference rule that can be used to derive new clauses from two given clauses. Given two clauses $\varphi \equiv \psi \vee P$ and $\vartheta \equiv \kappa \vee \neg P$, where $\psi$ and $\kappa$ are again clauses, then the resolution rule $\mathcal{R}$ derives the clause $\psi \vee \kappa$, formally $\varphi \wedge \vartheta \vdash_{\mathcal{R}} \psi \vee \kappa$. This simple inference rule is both sound and complete for deducing inconsistency (called refutation completeness), except when it is used with clauses in predicate logic, where it has to handle the situation where it can derive clause containing similar literals, only differing with respect their the universally quantified variables, such as $P(x) \vee P(y)$. Such a clause can be collapsed, by an operation called factoring. If this yields $P(x)$, resolution will be complete. Thus, resolution plus factoring is sound and refutation complete.

**Soundness:** Notion opposite in meaning to that of 'completeness'. If a collection of inference rules is sound, it is unable to derive anything from a logical formula $\varphi$ if that formula does not logically follow from that formula. Formally, if $\varphi \vdash \psi$ than $\varphi \vDash \psi$.

**Trace facility:** Component of an expert-system shell that allows following and inspecting the inference process.

**Utility:** Function value of a utility function.

**Utility function**: If $D_{V_i}$ represents the domain of a variable $V_i$, for $i = 1, \ldots, n$, then a utility function is defined as a mapping $u : D_{V_1} \times \cdots \times D_{V_n} \to \mathbb{R}$, where $u(v_1, \ldots, v_n) = c$, with $c \in \mathbb{R}$, has the informal meaning of a cost or benefit associated with a particular combination of values of variables.

## 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 article 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 $C_4H_9OH$ 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 call an *architecture*. These software issues are summarized in Section 2.1. Section 2.2 pays attention to expert

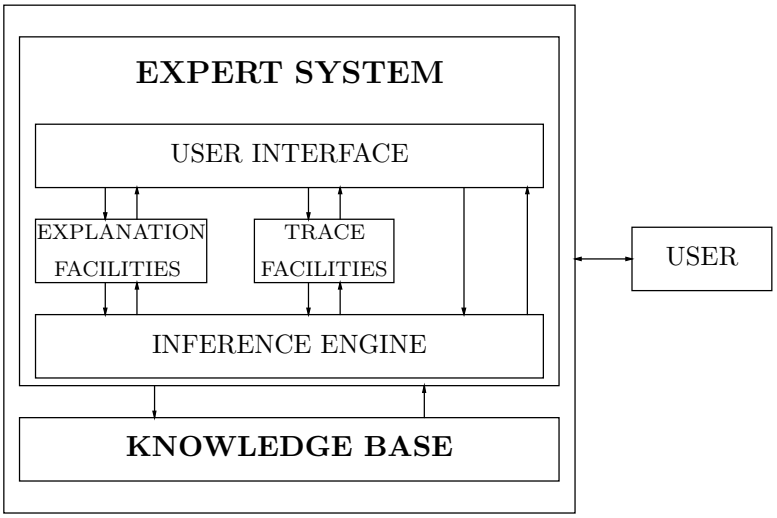


Figure 1: Global architecture of an expert system.

systems as programs solving particular types of problem.

## 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 issue 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 6.44: Section 1.4, in the article 'Logic in AI'.

### 3.1.1 Horn-clause Logic

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

$$\forall x_1 \cdots \forall x_m((A_1 \wedge \cdots \wedge A_n) \rightarrow B) \tag{1}$$

where $A_i, B$ are literals of the form $P(t_1, \ldots, t_q)$, 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 $Q \equiv (A_1 \wedge \cdots \wedge A_n) \rightarrow \perp$ be a query, then

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

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

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

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

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

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:

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

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 $F$ iff

  $$\mathrm{KB} \cup F \models S \tag{2}$$

  and $\mathrm{KB} \cup F \not\models \bot$, 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 $F$ iff the following *covering condition*

  $$\mathrm{KB} \cup S \cup K \models F \tag{3}$$

  is satisfied, where $K$ stands for *contextual knowledge*. In addition, it must hold that $\mathrm{KB} \cup S \cup C \not\models \bot$ (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 $S$, 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 $F$ iff

  $$\mathrm{KB} \cup S \cup F \not\models \bot \tag{4}$$

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.

### 3.1.2 Objects, Attributes and Values

Even though facts or observed findings can be represented in many different ways, in many expert systems facts are represented in an object-oriented fashion. This means that facts are described as properties, or *attributes*, of objects in the real world. Attributes of objects can be either multivalued, meaning that an object may have more than one of those properties at the same time, or singlevalued, meaning that values of attributes are mutually exclusive.

In logic, multivalued attributes are represented by predicate symbols, e.g.:

$$\text{Parent}(\text{John}, \text{Ann}) \wedge \text{Parent}(\text{John}, \text{Derek})$$

indicates that the 'object' John, represented as a constant, has two parents (the attribute 'Parent'): Ann and Derek, both represented by constants. Furthermore, singlevalued attributes are represented as function symbols, e.g.

$$\mathit{gender}(\text{John}) = \mathit{male}$$

Here, '*gender*' is taken as a singlevalued attribute, 'John' is again a constant object, and '*male*' is the value, also represented as a constant.

It is, of course, also possible to state general properties of objects. For example, the following bi-implication:

$$\forall x \forall y \forall z((\text{Parent}(x, y) \wedge \text{Parent}(y, z)) \leftrightarrow \text{Grandparent}(x, z))$$

defines the attribute 'Grandparent' in terms of the 'Parent' attribute.

Another typical example of reasoning about properties of objects is *inheritance*. Here one wishes to associate properties of objects with the classes the objects belong to, mainly because this yields a compact representation offering in addition insight into the general structure of a problem domain. Consider, for example, the following knowledge base KB:

$$\forall x(\text{Vehicle}(x) \rightarrow \text{HasWheels}(x))$$
$$\forall x(\text{Car}(x) \rightarrow \text{Vehicle}(x)$$
$$\forall x(\text{Car}(x) \rightarrow \mathit{number\text{-}of\text{-}wheels}(x) = 4)$$

Clearly, it holds that

$$\text{KB} \cup \{\text{Car}(\text{Bugatti})\} \models \mathit{number\text{-}of\text{-}wheels}(\text{Bugatti}) = 4$$

as the third rule expresses that as a typical property of cars. However, the knowledge base also incorporates more general properties of cars, such as:

$$\text{KB} \cup \{\text{Car}(\text{Bugatti})\} \models \text{Vehicle}(\text{Bugatti})$$

Now, given the fact that a car is a vehicle, we can now also conclude

$$\text{KB} \cup \{\text{Car}(\text{Bugatti})\} \models \text{HasWheels}(\text{Bugatti})$$

The example knowledge base discussed above can also be represented as a graph, called an object *taxonomy*, and is shown in Figure 2. Here ellipses indicate either classes of objects (Car and Vehicle) or specific objects (Bugatti). Solid arcs in the graph indicate that a class of objects is a subclass of another class of objects; a dashed arc indicates that the parent object is an element – often the term 'instance' is used instead – of the associated class of objects. The term 'inheritance' that is associated with this type of logical reasoning derives from the fact that the reasoning goes from the children to the parents in order to derive properties.

Describing the objects in a domain, usually but not always in a way resembling a taxonomy, usually with the intention to obtain a formal description of the terminology in a domain, is known as an *ontology*.

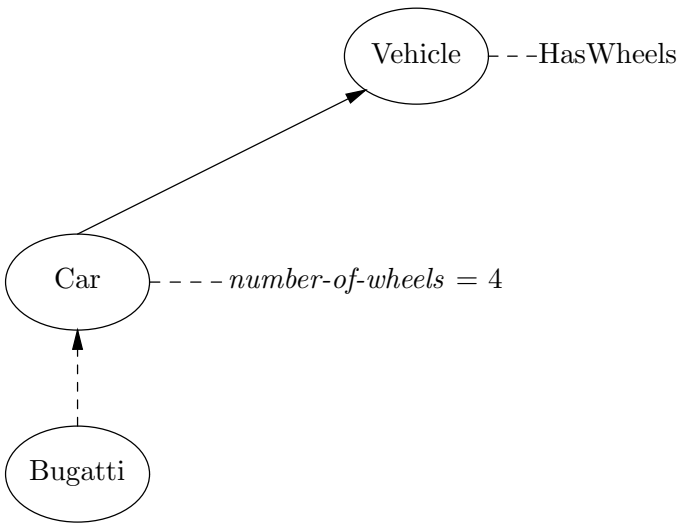


Figure 2: An object taxonomy.

### 3.2 Diagnostic Problem Solving

Above, the general features of knowledge representation and inference in expert systems were sketched. Most of the insight that has been gained in the field, however, concerns particular methods with associated knowledge to handle classes of problems. As said above, inference or reasoning methods can be used to implement problem-solving methods. A typical example is the diagnosis of disorders in patients or faults in equipment by diagnostic methods. Many different methods have been developed for that purpose. Three well-known diagnostic methods with their associated types of knowledge will be discussed in the following.

#### 3.2.1 Deductive Diagnosis

Most of the early expert systems, including DENDRAL and MYCIN, were based on expert knowledge concerning the relationships among classes expressed by rules. In the reasoning process these rules were subsequently used to classify cases into categories. This problem-solving method is known as *heuristic classification*, as most of the knowledge encoded in the rules is empirical or heuristic in nature rather than based on first principles. The form of the rules is:

$$(c_1 \wedge \cdots \wedge c_k \wedge \sim c_{k+1} \wedge \cdots \wedge \sim c_n) \rightarrow c$$

where $c_i$ is either a condition on input data or on a subclass. The rules are *generalized* rules, as conditions may be prefixed by a special negation sign $\sim$, called *negation by absence*. It represents a special case of the closed-world assumption (CWA, cf. Secion 6.44, Article 1.3 'Logic in AI'); a condition $\sim c_i$ only succeeds if there is at least one finding concerning the associated attribute. Formally:

$$\sim A(o, v) \equiv \exists x(A(o, x) \wedge x \neq v)$$

for object $o$ and value $v$, where $o$ and $v$ are constants. If the attribute $A$ represents a measurement or test, then negation by absence checks whether the test has been carried out, yielding a result different from the one specified.

Consider the following toy medical knowledge base KB:

$$\forall x((\text{Symptom}(x, \textit{coughing}) \wedge \sim \text{Symptom}(x, \textit{chest-pain}) \wedge \text{Sign}(x, \textit{fever})) \\ \quad \rightarrow \text{Disorder}(x, \textit{flu}))$$
$$\forall x((\textit{temp}(x) > 38) \rightarrow \text{Sign}(x, \textit{fever}))$$

Then it holds that:

$$\text{KB} \cup \{\textit{temp}(\text{John}) = 39, \text{Symptom}(\text{John}, \textit{coughing})\} \models_{\text{NA}} \text{Disorder}(\text{John}, \textit{coughing})$$

using negation by absence (NA). Note that $\text{Sign}(\text{John}, \textit{fever})$ is true, and may be viewed as a classification of the finding $\textit{temp}(\text{John}) = 39$; $\sim \text{Symptom}(\text{John}, \textit{chest-pain})$ holds due to negation by absence. Both rules in the knowledge base KB above are examples of heuristic classification rules.

### 3.2.2 Abductive Diagnosis

In abductive diagnosis, use is made of causal knowledge to diagnose a disorder in medicine or to determine faults in a malfunctioning device. Causal knowledge can be represented in many ways, but a rather convenient and straight-forward way to represent causal knowledge is by taking logical implication as standing for the causal relationship. Thus, rules of the form:

$$d_1 \wedge \cdots \wedge d_n \quad \rightarrow \quad f \tag{5}$$
$$d_1 \wedge \cdots \wedge d_n \quad \rightarrow \quad d \tag{6}$$

are obtained, where $d_i$ stands for a condition concerning a defective component or disorder; the conjunctions in (5) and (6) indicate that these conditions interact to either cause observable finding $f$ or another abnormal condition $d$ as effect. Sometimes uncertainty is added, usually represented in a non-numerical way as an assumption $\alpha$:

$$d_1 \wedge \cdots \wedge d_n \wedge \alpha_f \quad \rightarrow \quad f \tag{7}$$
$$d_1 \wedge \cdots \wedge d_n \wedge \alpha_d \quad \rightarrow \quad d \tag{8}$$

The literals $\alpha$ may be either assumed to be true or false, meaning that $f$ and $d$ are a possible, but not necessary, consequences of the simultaneous occurrence of $d_1, \ldots, d_n$.

An *abductive diagnosis* $S$ is now simply an abductive solution, where literals in $S$ are restricted to $d_i$'s and $\alpha$'s. The contextual knowledge may be extra conditions on rules which cannot be derived, but must be assumed and may act to model conditional causality. For simplicity's sake it is assumed here that $K$ is empty. The set of constraints $C$ may for instance consist of those findings $f$ which have not been observed, and are assumed to be absent, i.e. $\neg f$ is assumed to hold.

Consider, for example, the causal model with set of defects and assumptions:

$$\Delta = \{\textit{fever}, \textit{influenza}, \textit{sport}, \alpha_1, \alpha_2\}$$

and observable findings

$$\Phi = \{\textit{chills}, \textit{thirst}, \textit{myalgia}, \neg\textit{chills}, \neg\textit{thirst}, \neg\textit{myalgia}\}$$

'Myalgia' means painful muscles. The following knowledge base KB contains medical knowledge concerning influenza and sport, both 'disorders' with frequent occurrence:

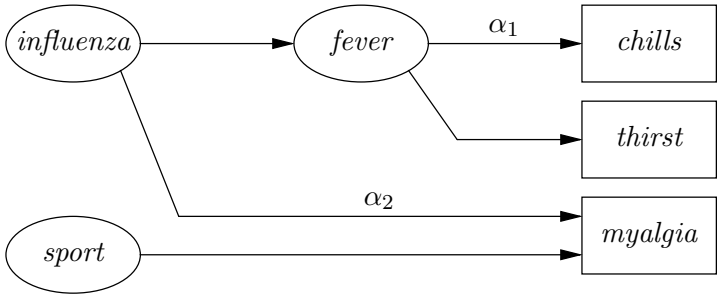


Figure 3: A knowledge base with causal relations.

$fever \wedge \alpha_1 \rightarrow chills$
$influenza \rightarrow fever$
$fever \rightarrow thirst$
$influenza \wedge \alpha_2 \rightarrow myalgia$
$sport \rightarrow myalgia$

For example, $influenza \wedge \alpha_2 \rightarrow myalgia$ means that influenza *may cause* myalgia; $influenza \rightarrow fever$ means that influenza *always causes* fever. For illustrative purposes, a causal knowledge base as given above is often depicted as a labelled, directed graph $G$, which is called a *causal net*, as shown in Figure 3. Suppose that the abductive diagnostic problem with set of facts

$F = \{thirst, myalgia\}$

must be solved. As constraints we take $C = \{\neg chills\}$. There are several solutions to this abductive diagnostic problem (for which the consistency and covering conditions are fulfilled):

$S_1 = \{influenza, \alpha_2\}$
$S_2 = \{influenza, sport\}$
$S_3 = \{fever, sport\}$
$S_4 = \{fever, influenza, \alpha_2\}$
$S_5 = \{influenza, \alpha_2, sport\}$
$S_6 = \{fever, influenza, sport\}$
$S_7 = \{fever, influenza, \alpha_2, sport\}$

Note that $S = \{\alpha_1, \alpha_2, fever, influenza\}$ is incompatible with the constraints $C$.

### 3.2.3 Consistency-based Diagnosis

In consistency-based diagnosis, in contrast to abductive diagnosis, the malfunctioning of a device is diagnosed by using mainly knowledge of the normal structure and normal behavior of the components of a device or system. For each component $\text{COMP}_j$ its normal behavior is described by logical implications of the following form:

$$\forall x((\text{COMP}_j(x) \wedge \neg\text{Ab}(x)) \rightarrow \text{Behavior}_j(x))$$

The literal $\neg\text{Ab}(x)$ expresses that the behavior associated with the component only holds when the assumption that the component is not abnormal, i.e. $\neg\text{Ab}(c)$, is true for component $c$. Sometimes knowledge of abnormal behavior is added to implications of the form above, having the form:

$$\forall x((\text{COMP}_j(x) \wedge \text{Ab}(x)) \rightarrow \text{Behavior}_j(x))$$

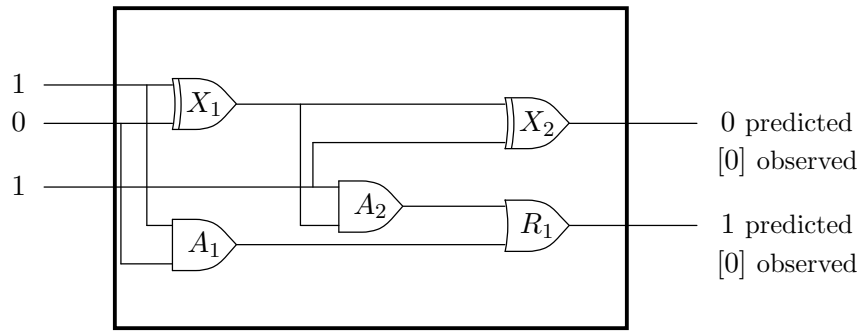


Figure 4: Full adder.

These may result in a reduction in the number of possible diagnoses to be considered. Logical behavior descriptions of the form discussed above are part of a *system description*, referred to by SD. In addition to the generic descriptions of the expected behavior of components, a system description also includes logical specifications of how the components are connected to each other (the structure of the system or device), and the names of the actual components making up the system or device. The system description is now taken as the knowledge base KB of an expert system. Problem solving basically amounts to adopting particular assumptions about every $\text{COMP}_j(c)$, either whether $\text{Ab}(c)$ is true or false. This sort of reasoning is called *assumption-based* or *hypothetical reasoning*.

Consider the logical circuit depicted in Figure 4, which represents a full adder, i.e. a circuit that can be used for the addition of two bits with carry-in and carry-out bits. The components $X_1$ and $X_2$ represent exclusive-OR gates, $A_1$ and $A_2$ represent AND gates, and $R_1$ represents an OR gate.

The system description KB consists of the following axioms:

$$\begin{aligned}
\forall x(\text{ANDG}(x) \wedge \neg\text{Ab}(x) &\rightarrow out(x) = and(in1(x), in2(x))) \\
\forall x(\text{XORG}(x) \wedge \neg\text{Ab}(x) &\rightarrow out(x) = xor(in1(x), in2(x))) \\
\forall x(\text{ORG}(x) \wedge \neg\text{Ab}(x) &\rightarrow out(x) = or(in1(x), in2(x)))
\end{aligned}$$

which describe the (normal) behavior of each individual component (gate), and

$$\begin{aligned}
out(X_1) &= in2(A_2) \\
out(X_1) &= in1(X_2) \\
out(A_2) &= in1(R_1) \\
in1(A_2) &= in2(X_2) \\
in1(X_1) &= in1(A_1) \\
in2(X_1) &= in2(A_1) \\
out(A_1) &= in2(R_1)
\end{aligned}$$

which gives information about the connections between the components, i.e. information about the normal structure, including some electrical relationships. Finally, the actual gates are defined:

$$\begin{aligned}
&\text{ANDG}(A_1) \\
&\text{ANDG}(A_2)
\end{aligned}$$

XORG($X_1$)
XORG($X_2$)
ORG($R_1$)

Appropriate axioms of a Boolean algebra are also assumed to be available.

Now, it assumed that the following observations have been made:

$$F = \{in1(X_1) = 1, in2(X_1) = 0, in1(A_2) = 1, out(X_2) = 0, out(R_1) = 0\}$$

Note that $out(R_1) = 1$ is predicted using the model of normal structure and behavior in Figure 4, which is in contrast with the observed output $out(R_1) = 0$. This indicates that the device is malfunctioning. This is also verified by noting that when assuming all components to behave normally, i.e. $S = \{\neg Ab(c) \mid c \in \text{COMPS}\}$, it follows that

$$\text{KB} \cup S \cup F$$

is inconsistent. This confirms that some of the output signals observed differ from those expected under the assumption that the circuit is functioning normally.

Diagnosing the problem simply consists of assuming particular components to be abnormal (Ab($c$) is true for those components), and checking whether the result is still inconsistent. If it is not, a diagnosis has been found. So, a *consistency-based diagnosis* is a consistency-based solution $S$ consisting of a conjunction of Ab literals, one for every component.

Consider again the example above. Here,

$$S = \{\text{Ab}(X_1), \neg\text{Ab}(X_2), \neg\text{Ab}(A_1), \neg\text{Ab}(A_2), \neg\text{Ab}(R_1)\}$$

is a consistency-based diagnosis as

$$\text{KB} \cup S \cup F \nvDash \bot$$

Note that $X_1$ is possibly faulty, as assuming it to be abnormal yield no output for this components, so $A_2$ will not receive the necessary 1 to make the output of $R_1$ inconsistent with the observations. Of course, the are many other solutions as well.

### 3.3 Configuring and Design

*Configuring* a device or system amounts to making choices of which and how components need to be combined to yield the requested device or system. *Designing* a system involves even more degrees of freedom than configuring a system, but is related to that task. Designing and configuring have in common that (partial) results must fulfill certain contraints and requirements. A simple example of a design problem is shown in Figure 5.

In this design problem, filling in the box with the components of a full adder as in Figure 4 would in fact be one way to obtain the requested functional behavior, but there are other possibilities as well. Let KB be a knowledge base consisting of generic logical descriptions of the functional behavior of the components which may be used to design a system. Let the set of observed findings $F$ be equal to the set $O$ of expected outputs; the context $K$ of the problem is assumed to be equal to the set of inputs $I$. Finally, let $S$ stand for the resulting design solution, consisting of the actual components and interconnections between components. We call $S$ a *potential design solution* iff

$$\text{KB} \cup S \cup K \vDash F$$

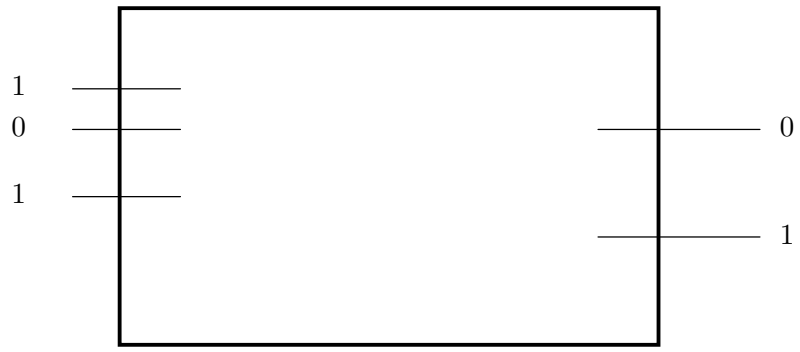


Figure 5: Functional requirements of a circuit.

i.e. the covering condition holds. Using the consistency condition with constraints $C$ allows us to check the solution. The result is then an *acceptable design solution* $S$. In other words, $S$ is an inductive solution to the design problem.

In the case of the example in Figure 5, the knowledge base KB could for example consist of the following formulae:

$$\begin{aligned} \forall x(\text{ANDG}(x) &\rightarrow out(x) = and(in1(x), in2(x))) \\ \forall x(\text{XORG}(x) &\rightarrow out(x) = xor(in1(x), in2(x))) \\ \forall x(\text{ORG}(x) &\rightarrow out(x) = or(in1(x), in2(x))) \end{aligned}$$

which describe the behavior of each type of component. Several of the components may be used, sometimes more than once, in designing the system $Y$.

For the inputs and outputs it holds that:

$$K = \{in1(Y) = 1, in2(Y) = 0, in3(Y) = 1\}$$

and

$$F = \{out1(Y) = 0, out2(Y) = 0\}$$

A potential design solution $S$ may include the following elements:

$$\begin{aligned} &\text{ANDG}(A_1) \\ &\text{ANDG}(A_2) \\ &\text{XORG}(X_1) \\ &\text{XORG}(X_2) \\ &\text{ORG}(R_1) \end{aligned}$$

$$\begin{aligned} in1(Y) &= in1(X_1) \\ in2(Y) &= in2(X_1) \\ in3(Y) &= in2(A_1) \\ out1(Y) &= out(X_2) \\ out2(Y) &= out(R_1) \\ out(X_1) &= in2(A_2) \\ out(X_1) &= in1(X_2) \end{aligned}$$

$$
\begin{aligned}
out(A_2) &= in1(R_1) \\
in1(A_2) &= in2(X_2) \\
in1(X_1) &= in1(A_1) \\
in2(X_1) &= in2(A_1) \\
out(A_1) &= in2(R_1)
\end{aligned}
$$

Obviously, this design solution is able to derive the expected output $F$ given the knowledge base KB and the input $K$.

One of the major problems of automatic design is to cut down on the number of possibilities that must be considered. Even for the simple problem discussed in the example above, finding an acceptable design solution will be intractable. By using additional domain knowledge, it may be possible to focus the reasoning process.

### 3.4 Uncertainty

Up until now, it has been assumed that in representing and solving a problem in a domain dealing with uncertainty is not of major importance. As this does not hold for many problems, the possibility to represent and reason with the uncertainty associated with a problem is clearly of significance. There have been a number of early attempts where researchers have augmented rule-based, logical methods with uncertainty methods, usually different from probability theory, although sometimes also related. However, those methods are now outdated, and have been replaced by methods which take probability theory as a starting point. In the context of expert systems, in particular the formalism of Bayesian belief networks has been successful. Expert systems built using this formalism are called *probabilistic expert systems* or *normative expert systems*.

A *Bayesian belief network* $\mathcal{B} = (G, \Pr)$, also called *causal probabilistic network*, is a directed acyclic graph $G = (V(G), A(G))$, consisting of a set of nodes $V(G) = \{V_1, \ldots, V_n\}$, called *probabilistic nodes*, representing discrete random variables, and a set of arcs $A(G) \subseteq V(G) \times V(G)$, representing causal relationships or correlations among random variables. Consider Figure 6, which shows a simplified version of a Bayesian belief network modeling some of the relevant variables in the diagnosis of two causes of fever. The presence of an arc between two nodes denotes the existence of a direct causal relationship or other influences; absence of an arc means that the variables do not influence each other directly. The following knowledge is represented in Figure 6: variable 'FL' is expressed to influence 'MY' and 'FE', as it is known that flu causes myalgia (muscle pain) and fever. In turn, fever causes a change in body temperature, represented by the random variable TEMP. Finally, pneumonia (PN) is another cause of fever.

Associated with a Bayesian belief network is a joint probability distribution Pr, defined in terms of conditional probability tables according to the structure of the graph. For example, for Figure 6, the conditional probability table

$$\Pr(\text{FE} \mid \text{FL}, \text{PN})$$

has been assessed with respect to all possible values of the variables FE, FL and PN. In general, the graph associated with a Bayesian belief network mirrors the (in)dependences that are assumed to hold among variables in a domain. For example, given knowledge about presence or absence of fever, neither additional knowledge of flu nor of pneumonia is able to

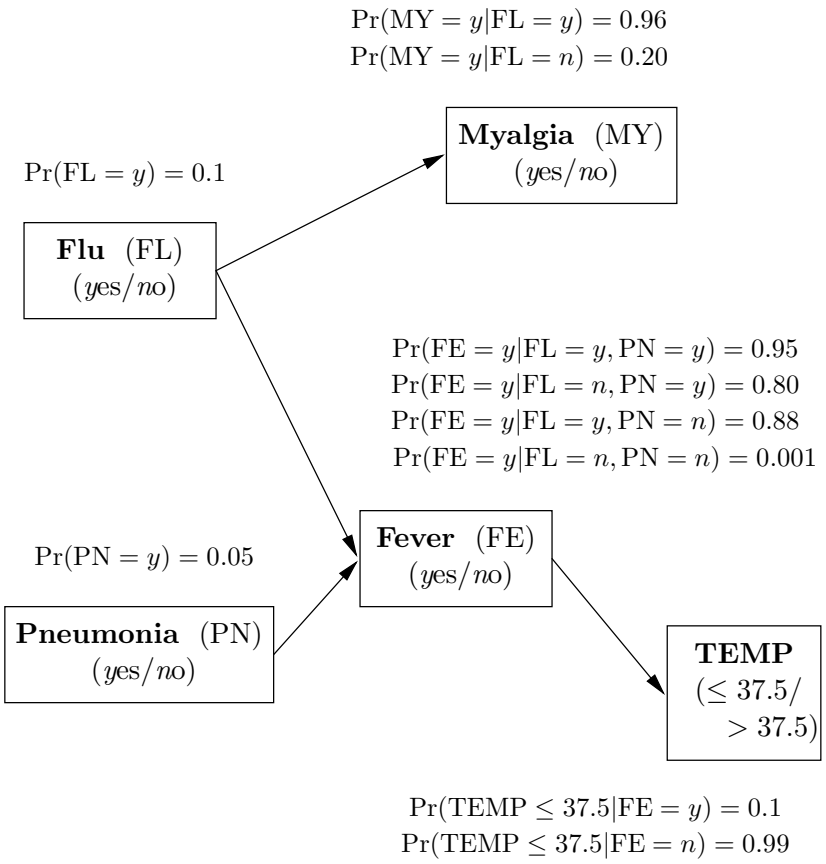


Figure 6: Bayesian network $\mathcal{B}=(G,\Pr)$ with associated joint probability distribution Pr (only probabilities $\Pr(X=y\,|\,\pi(X))$ are shown, as $\Pr(X=n\,|\,\pi(X))=1-\Pr(X=y\,|\,\pi(X))$).

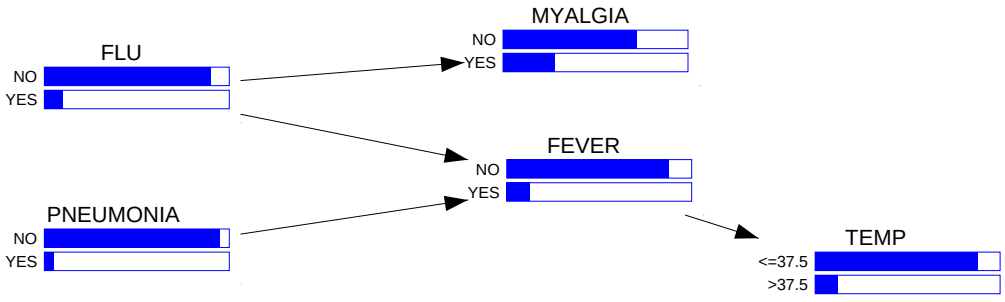


Figure 7: Prior marginal probability distributions for the Bayesian belief network shown in Figure 6.

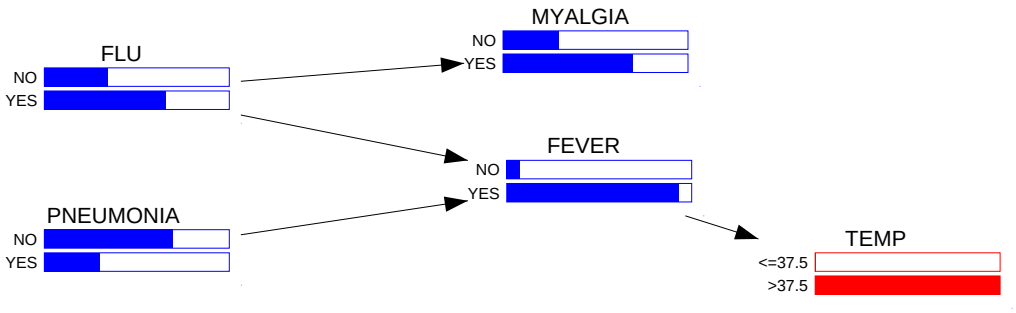


Figure 8: Posterior marginal probability distributions for the Bayesian belief network after entering evidence concerning body temperature. Note the increase in probabilities of the presence of both flu and pneumonia compared to Figure 7. It is also predicted that it is likely for the patient to have myalgia.

influence the knowledge about body temperature, since it holds that TEMP is conditionally independent of both PN and FL given FE.

For a joint probability distribution defined in accordance with the structure of a Bayesian network, it, therefore, holds that:

$$\Pr(V_1, \ldots, V_n) = \prod_{i=1}^{n} \Pr(V_i \mid \pi(V_i))$$

where $V_i$ denotes a random variable associated with an identically named node, and $\pi(V_i)$ denotes the parents of that node. As a consequence, the amount of probabilistic information that must be specified, exponential in the number of variables in general when ignoring the independencies represented in the graph, is greatly reduced.

By means of special algorithms for probabilistic reasoning – well-known are the algorithms by Pearl [13] and by Lauritzen and Spiegelhalter [11] – the marginal probability distribution $\Pr(V_i)$ for every variable in the network can be computed; this is shown for the fever network in Figure 7. In addition, a once constructed Bayesian belief network can be employed to enter and process data of a specific case, i.e. specific values for certain variables, like TEMP, yielding an updated network. Figure 8 shows the updated Bayesian network after entering evidence about a patient's body temperature into the network shown in Figure 6. Entering evidence in a network is also referred to as *instantiating* the network. The resulting probability distribution of the updated network, $\Pr^E(V_i)$, which is a marginal probability distribution of the probability distribution $\Pr^E$, is equal to the posterior of the original probability distribution of the same variable, conditioned on the evidence $E$ entered into the network:

$$\Pr^E(V_i) = \Pr(V_i \mid E)$$

Bayesian belief networks have also been related to logic by so called *probabilistic Horn clauses*. This formalism offers basically nothing else then a recipe to obtain a logical specification of a Bayesian belief network. Reasoning with probabilistic Horn clauses is accomplished by logical abduction; the axioms of probability theory are used to compute an updated probability distribution.

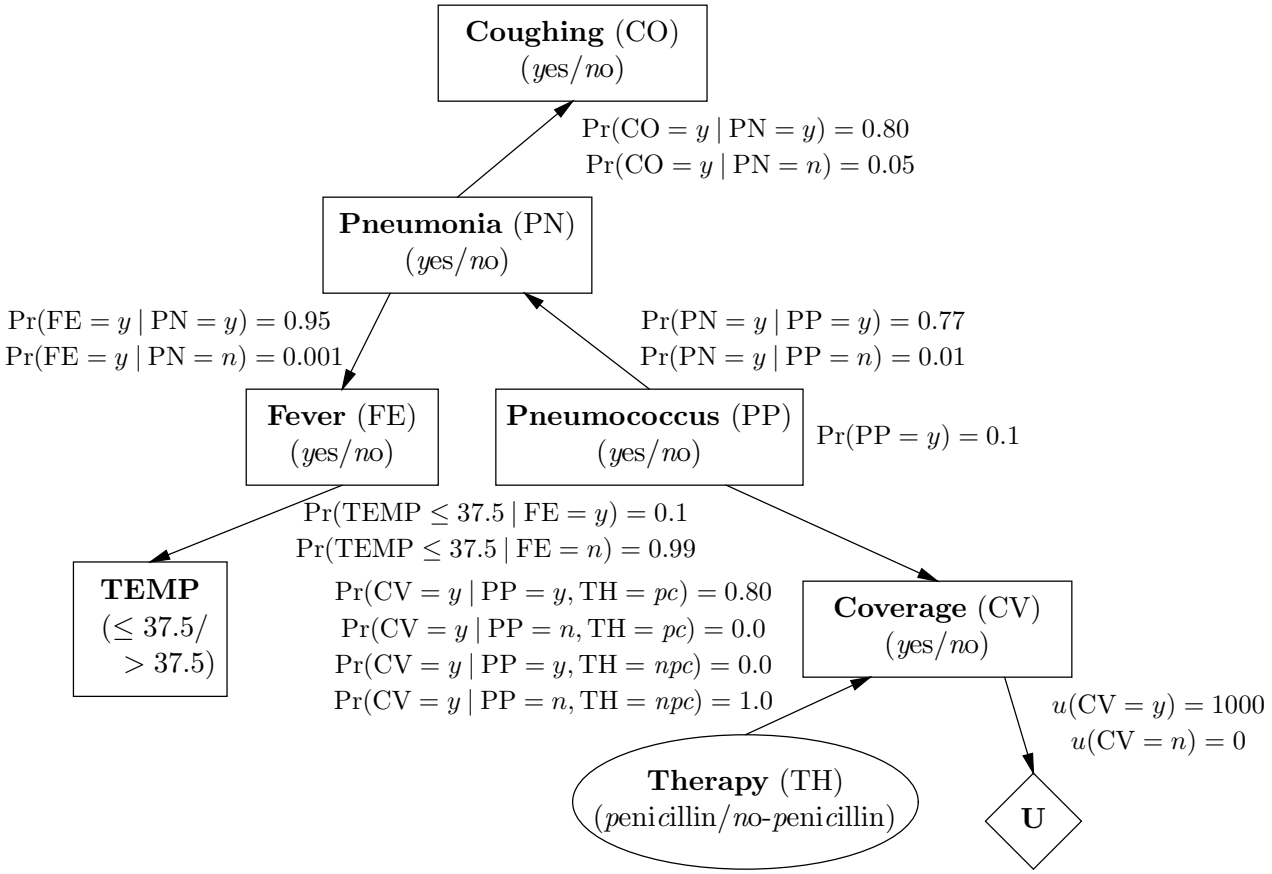


Figure 9: Influence diagram.

### 3.5 Decision Making

The Bayesian-network formalism provides only for probabilistic reasoning. For making decisions, certain extensions to the Bayesian-network formalism are required, as offered, among others, by the influence-diagram formalism. Like a Bayesian network, an *influence diagram* is an acyclic directed graph $G = (V(G), A(G))$, except that in addition to probabilistic nodes $O(G) \subseteq V(G)$, two additional node types are distinguished: *decision nodes* $D(G) \subseteq V(G)$ and a *value node* $U \in V(G)$, where $O(G) \cap D(G) = O(G) \cap \{U\} = D(G) \cap \{U\} = \varnothing$. A probabilistic node is denoted by means of a box, a decision node is indicated by means of an ellipse, and a value node is indicated by a diamond.

As holds for Bayesian networks, arcs between probabilistic nodes denote direct causal relationships or correlations. An example of such an uncertain influential relationship in Figure 9 is the arc between 'PP' (presence of the bacterium Pneumococcus in a sputum culture) and 'PN' (presence of pneumonia in a patient). Pneumococcus is a very frequent cause of pneumonia, which explains both the presence of an arc between the two nodes and also the direction of the arc. In turn, pneumonia gives rise to a number of signs and symptoms, such as fever (FE) and coughing (CO). The arcs from the node PN to the nodes FE and CO reflect that causal knowledge.

Incoming arcs to a decision node indicate information that is required for the purpose of making a decision; therapy (TH) is the decision node in Figure 9. In the example influence

diagram it is assumed that therapy selection can be done without taking any information but the effect of therapy into account. However, if particular variables, such as TEMP and CO are instantiated, then they are handled as if there is an information arc from these nodes to the decision node TH. In this example there is just a single decision node, but if there is more than one the order of decision nodes in an influence diagram expresses the order in which the decisions are made in the domain. Usually, it is assumed that this order is complete, although, in principle, a partial order can be employed to indicate alternative sequences of decisions.

In order to select the most optimal sequence of decisions, assessments of these decisions with respect to other, partially uncertain, information must be available. These assessments, called *utilities*, are represented by the value node. A utility is a quantitative measure of the strength of preferences for particular outcomes. Incoming arcs of the value node indicate variables on which utilities must be defined, i.e. utilities can be expressed as a *utility function*:

$$u : C_{\pi(U)} \rightarrow \mathbb{R}$$

where $C_{\pi(U)}$ denotes the set of possible instantiations of parents of the value node $U$. In Figure 9 it is expressed that any decision concerning the best treatment for a patient with pneumonia should only be based on the coverage, i.e. the potential to kill in vivo, of the causative bacterium Pneumococcus by Penicillin. In this simplified example, many factors that normally would have been included, such as side-effects of antibiotic treatment and costs of the treatment, have been omitted.

As for Bayesian networks, the graph representation only denotes the qualitative relationships between variables; it must be supplemented with a quantitative representation to obtain an influence diagram. Example local probability distributions and a utility function for the domain of antibiotic treatment of pneumonia are shown in Figure 9. Various algorithms to evaluate influence diagrams are available, e.g. the algorithms proposed by Shachter [18] and Cooper [7]; the result of evaluation is an optimal sequence of decisions, a *decision strategy*, obtained by computation of the maximum expected utility $\bar{u}$ of every decision $D \in D(G)$ given particular evidence $E$:

$$\bar{u}(D \mid E) = \max_{d \in D} \sum_{c \in C_{\pi(U)}} u(c) \Pr(c \mid d, E)$$

where possibly $D \in \pi(U)$, and the evidence $E$ includes previous decisions.

# 4 Knowledge Engineering

Gathering, analyzing and modeling knowledge are activities necessarily undertaken when developing an expert system. The knowledge can be either (semi)automatically extracted from databases and datasets, using machine-learning techniques, or be obtained by consulting literature and experts in a given domain. Even though learning knowledge automatically from data is appealing, in many cases learning of knowledge appears impossible without at least substantial input from an expert in the domain. Actually modeling knowledge in building expert systems is therefore as important now as it was in the days of DENDRAL and MYCIN.

Whereas in the early days of expert systems, people had a tendency to translate collected domain knowledge almost straight into a set of rules, modern knowledge-engineering methodologies put emphasis on systematic knowledge base development, using a number of

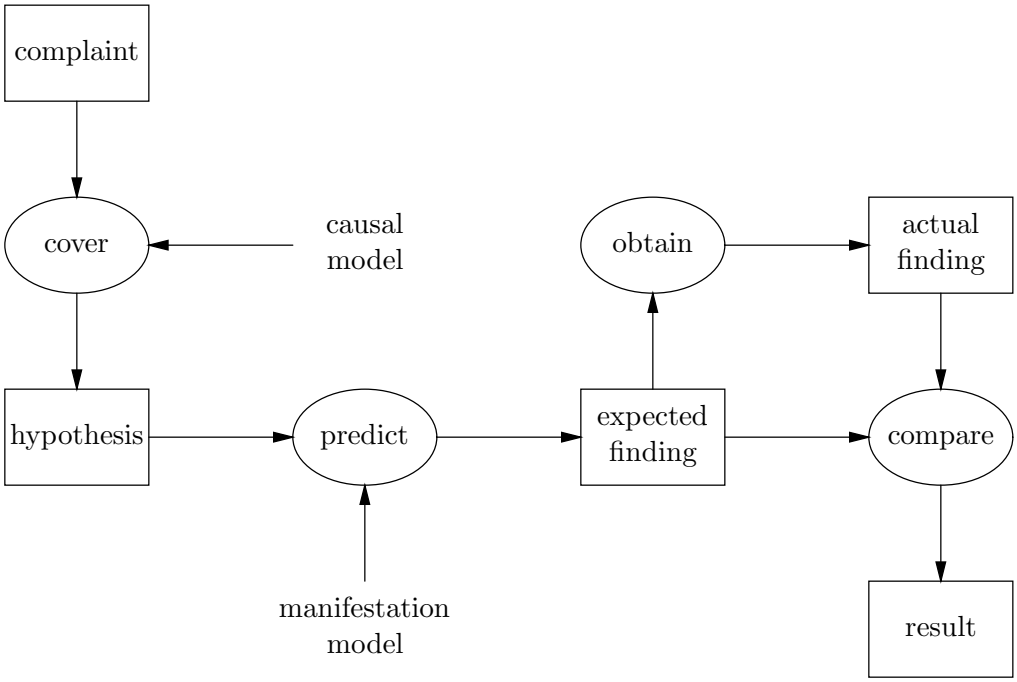


Figure 10: CommonKADS inference structure, annotated with domain knowledge (based on [17]).

intermediate results before putting it all together as a working expert system. A well-known example of such a methodology is CommonKADS.

CommonKADS takes knowledge base development as a sequence of interrelated modeling activities. The aim is to develop a so-called *design model* of an expert system. This consists of two separate but linked parts:

- a *communication model*, which describes in what way and how an expert system will communicate with its environment, consisting for example of the user, particular equipment, and so on;
- a *knowledge model*, which describes the gathered knowledge of a domain in addition to how that knowledge may be employed to solve problems in the domain.

As the major contribution of CommonKADS lies in the concept of knowledge model, the communication model will not be discussed any further.

The notion of knowledge model in CommonKADS has a broad meaning, as it does not only involve all aspects associated with the types of knowledge used in dealing with certain problems, but also the types of reasoning to be employed. As the methodology attempts to provide generic, but still precise guidelines, there is much emphasis on schemata that can be used to model related domains. It is best to illustrate the CommonKADS methodology with principles introduced in Section 3.1, as there are many similarities between the formal methods reviewed in that section and the CommonKADS approach to knowledge modeling.

CommonKADS makes a distinction between *domain knowledge*, *inference knowledge* and *task knowledge*. For example, in diagnostic problem solving, causal knowledge with an associated indication of those elements of the causal knowledge which may be taken as part of a

hypothesis $S$, and elements which correspond to findings that may be actually observed in the real world would be classified as domain knowledge. Definition (3) of an abductive diagnosis would be looked upon as inference knowledge, as it describes which hypotheses (for example diagnostic hypotheses) would be accepted as a (diagnostic) solution. Finally, task knowledge in this case would concern procedures that describe how hypotheses can be generated and tested against observed findings, using domain and inference knowledge. An example of a diagram of a diagnostic inference model is shown in Figure 10.

Rather than offering a set of formal techniques to specify a knowledge model, most of CommonKADS concerns semiformal techniques, which often have no precise semantics. Both diagrammatic and textual specifications are used to represent a knowledge model. However, there have been several attempts to offer a formal underpinning of the approach in a manner similar to that discussed in Section 3.1.

Even though the CommonKADS methodology suggests that it covers most types of expert system development, this is not the case for expert systems based on probability theory and decision theory, which can be viewed upon as one of the omissions of the approach.

Important final aspects of expert-system development are the evaluation and validation of the resulting system. This is done by comparing the system with the original requirement of the system (validation) and by studying the problem-solving behavior of the system for problem cases (*laboratory test*) and in the working environment (*field test*).

# 5 Conclusions

In this article, the most important topics in the field of expert systems have been discussed. The central issue of expert-system development is knowledge modeling, and many different formalisms are available today to model the features of a problem domain accurately. As a consequence, it is not always easy to decide whether a particular approach to developing an expert system is the right one.

# Bibliography


[1] 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 describes.]

[2] 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 many different aspects of the MYCIN project.]

[3] 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.]

[4] 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.]

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

[6] 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.]

[7] 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.]

[8] 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.]

[9] 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.]

[10] 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.]

[11] 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 a new methods in 1987, now widely used, for probabilistic reasoning with Bayesian belief networks.]

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

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

[14] 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.]

[15] 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.]

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

[17] 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.]

[18] 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.]