Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

P.J.F. Lucas

Knowledge representation and inference
in rule-based systems

Computer Science/Department of Software Technology Report CS-R8613 April

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

Knowledge Representation and Inference in Rule-Based Systems

P.J.F. Lucas
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

In this paper a review is presented of various approaches in representing and applying human knowledge in expert systems, in particular in rule-based systems. The paper also provides an introduction to some equivalent methods of representation. Some emphasis is put on low-level operations and also on inference procedures that are applied in extracting useful knowledge from a knowledge base. This investigation is partly based on work done in the design and the implementation of the DELFI-2 system at Delft University of Technology and recently at the Centre for Mathematics and Computer Science. It has particularly been influenced by concepts from logic programming.
Key Words & Phrases: expert systems, knowledge-based systems, inference.
I.2.1, I.2.4

1. Introduction

Expert system building tools, also called expert system shells, are a result of progress in the field of Artificial Intelligence in the design of practical software tools to be used for the efficient solution of problems, that are generally hard to solve by other means. In the last two decades, a shift is observed from research directed at the design of general purpose problem solving methods towards investigations aimed at the design of representation formalisms. Earlier systems often lacked sufficient power for dealing with complex real-life problems. The representation of human knowledge in the computer turned out to be a key issue in expert system research. Although much effort has been spent on this issue, it is also becoming clear that efficient means for inference remain to be considered. Thus, work in Artificial Intelligence developed in a similar way as may be observed in other parts of computer science: emphasis changed from imperative or procedural methods, which predominated the design of programming languages for a long time, to descriptive techniques instead. These changes led to the design of software tools that are able to apply a problem description more or less intelligently in solving a certain class of problems. A major objective in expert system research is to keep description and use of knowledge completely separated; the former being developed by the knowledge engineer, and the latter by the computer scientist. Still, there is a long way to go before reaching such a complete separation, because it often turns out that general purpose languages are more flexible that the rather restricted expert system shells. Nevertheless, expert system shells have been successful in dealing with certain problems [1].

What has been established are expert system shells that provide means for knowledge representation and a set of inference methods to supply the user with advice. Thus, most current expert systems are composed of the following two parts [2]:
- a knowledge base, containing problem specific knowledge;
- a consultation program, that is essentially problem independent.

In earlier expert system shells, the knowledge base often had to be specified in the same programming language as applied in writing the consultation program; such was the case in the EMYCIN system, where the problem specific knowledge had to be represented as LISP s-expressions. In the more recently developed systems this is less often true. For example, in the DELFI-2 system knowledge is represented in a symbolic specification language, while the consultation system has been developed in the Pascal programming language [2].

Report CS-R8613
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

The consultation system typically comprises certain subsystems, of which the inference engine, essentially a knowledge interpreter, is of crucial importance. This subsystem incorporates one or more control strategies that apply the knowledge stored in the knowledge base, to derive new information. In addition, there are often facilities available for explanation and debugging purposes. The explanation facility is primarily present for increasing the confidence of the user in a once built expert system. The building process itself is supported by means of debugging and tracing facilities. A standard user interface is provided to facilitate building specific applications. The various components of such an expert system are shown in figure 1.

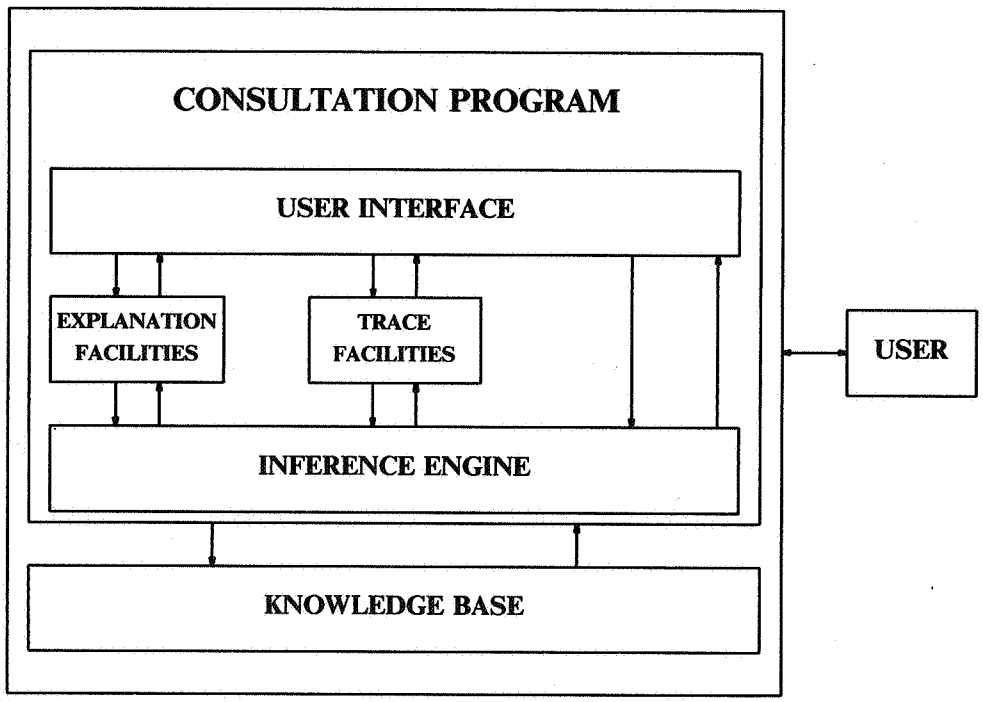

Figure 1. Global architecture of an expert system shell

At present, there are many expert system building tools available with a variety of approaches in both knowledge representation and inference. There is a unmistakable trend towards building tools that provide the user with an even larger variety of knowledge representation formalisms and inference techniques, often integrated into one tool. Most systems offer additional flexibility by allowing the implementer of the expert system to escape to a specific programming language environment (such as LISP), whenever the built-in facilities appear inadequate for a particular application.

2. Representing Human Knowledge

Knowledge representation is one of the foremost topics in expert system research. This is partly caused by the emphasis that is placed on the descriptive aspects of expert systems. On the other hand, as we shall see, research dealing with control issues is not really less important to the researcher, but it is to the knowledge engineer, who only is concerned with the specific tools.

There are two prerequisites for a knowledge representation scheme before any claim of its success can be made [3]:

- the scheme should be flexible, capable of representing a large variety of knowledge;

- it must be simple enough to enable translation to natural language, thus making it easier to understand for the user.

There are two frequently applied techniques for encoding knowledge. In practice, most systems use production rules, with each rule having a condition-part and an action-part, as the principal technique for knowledge representation. Another approach is based on the representation of knowledge into structured objects, also called frames or prototypes. Although the last method appears to be more flexible, because it allows constructing an expert system in a modular hierarchical fashion, on many occasions production rules have been shown to be superior for building specific applications. Production rules often appeal to the co-operating domain-expert.

2.1. Simple production rule formalisms
A production rule can be defined as follows [3]:

```
if <antecedent> then <consequent> fi

<antecedent> ::= <clause> { and <clause> }

<clause>     ::= <condition> { or <condition> }

<consequent> ::= <conclusion> { and <conclusion> }
```

There are many techniques in use for specifying conditions and conclusions within the formalism of production rules. The most simple one being both conditions and conclusions written down as propositions, for example:

```
if
  (patient has a fever) and
  (patient has colicky pain)
then
  (diagnosis may be bile duct stones)
fi
```

Both conditions and conclusions within parentheses are propositions. It must be stressed that production rules, although very similar to implications in logic, are treated somewhat differently from logical implications. For example, in this case "diagnosis may be bile duct stones" is added to a so-called set of facts. The implicational notation for production rules has certain advantages, owing to the difference in treatment of conditions and conclusions by the inference engine. The activity of an inference engine may be regarded as finding matches between conditions and conclusions of production rules, producing a rule connection graph. In order to make this possible, rule conditions and conclusions have to be distinguished. Thus, the well-known equivalence between the following two logical formulas:

```
a and b → c
not(a and b) or c
```

is not of much use in expert systems. In expert systems, a, b on the one hand and c on the other hand are treated differently.

Generally, in rule-based expert systems the following three types of knowledge are distinguished: production rules constituting a rule base, facts added to a set of facts when established, and goals that activate the control strategy in some cases and in others constitute a termination criterion.

Using propositions as a knowledge representation scheme is a rather simple and restrictive method, and it is probably difficult to develop serious expert systems with this scheme. More flexible knowledge representation schemes offer the knowledge engineer variables and predicates and actions that perform certain tests on variables and constants. Using this representation scheme, the foregoing rule might look as follows:

```
if
   Same(fever,YES) and
   Same(pain,COLICKY)
then
   Assert(diagnosis,BILE-DUCT-STONES)
fi
```

In this rule, variables are indicated by lower-case and constants by upper-case characters. The predicate "Same" performs a test on the variables "fever" and "pain" and on the constants "YES" and "COLICKY" for equality. The action "Assert" assigns the constant value "BILE-DUCT-STONES" to the variable "diagnosis". Thus, if before the action "Assert" is executed, the set of facts looks like

{fever=YES,pain=COLICKY}

then after execution we have:

{fever=YES,pain=COLICKY,diagnosis=BILE-DUCT-STONES}

In general, the set of facts is defined as:

$$\{x_1=C_1,\ldots,x_n=C_n\}$$

with each member $x_i = C_i$ (i = 1,..,n) being an established fact.

The variable "diagnosis" might have been a goal variable, in which case it was initially a member of the goal set.

More flexibility can be gained if we add an additional method for organizing knowledge in structured objects. Within rule-based systems, objects are mainly used for organizing the production rules, and collecting the variables in these rules under one heading. Variables within objects are also called attributes or parameters. Structured objects form the basis of frame-based expert systems if the relationship between objects is of major importance.

2.2. Rule base organization and objects

The introduction of objects leads to a functional separation of a knowledge base into production rules for representing heuristic expert knowledge and structured objects which store descriptive information. The object descriptions might be rather extensive, not only containing attribute names, but also information for the user interface, constraints on user input, information on which and how attributes should be derived. The set of goals is incorporated into this object description: a goal simply is an attribute with a special *goal* label. The integration of both schemes, rules and structured objects, into a single knowledge base, imposing some kind of hierarchical organization upon the rule set, is the result of recent investigations in the field of Artificial Intelligence. Knowledge base modularization appears to be a key issue, because both the construction and consultation of a knowledge base, are facilitated by the provision of multiple organizational levels in the knowledge base. This view leads to a rule base that is divided into separate subsets, each related to a different object.

Each attribute of an object has certain information associated with it, in particular a name, a

translation of this name used when communicating with the user, a prompt, i.e. a potential question for user input and constraints on legal input. A formalism for object description can be based upon first-order predicate calculus.

A description of an attribute based upon logic, may be the following specification [4]:

```
attribute(A) and
name(A,N) and
translation(A,T) and
prompt(A,P) and
constraints(A,V1,...,Vn) and
values(A,x1,...,xm)
```

This formula contains constants, indicated by upper-case and variables by lower-case characters. The relationships between the attribute constant "A" and other constants and variables are described by means of predicates.

If we introduce a standard predicate "equals", a simple transformation of the previous specification leads to the next attribute description:

```
attribute(A) and
equals(name(A),N) and
equals(translation(A),T) and
equals(prompt(A),P) and
equals(constraints(A),V1,...,Vn) and
equals(values(A),x1,...,xm)
```

We use functions, having the attribute constants as a domain, to describe the attribute components. The following notation is used more often:

```
attribute: A
  name: N;
  translation: T;
  prompt: P;
  constraints: (V1,...,Vn);
  values: (x1,...,xm)
end
```

The following example illustrates how this descriptive formalism can be used to specify an attribute, called "complaints", of a patient with liver disease:

```
attribute: complaints
  name: complaints;
  translation: the complains of the patient;
  prompt: What complaints does the patient have;
  constraints: (anorexia, fever, jaundice, nausea);
  values: (fever)
end
```

In this example one of the variable values $x_1,...,x_m$ ($m \geq 0$) has been substituted by the constant value "fever".

A frame is similar to our description of objects, but has additional information concerning links to other objects. If the links bear semantic information, the objects and their connecting links form a so-called semantic net. An often used link in a semantic net is the *ISA-link* that imposes a hierarchical organization to the objects, called a *taxonomy.*

We next discuss how objects and attributes are used within production rules, such as for example in the DELFI-2 system but also in many precursors of this system, in particular the EMYCIN system on which DELFI-2 has been based.

The conditions and conclusions in production rules are syntactically defined as follows:

```
<condition>  ::= <predicate> <object> <attribute> { <constant> }

and

<conclusion> ::= <action> <object> <attribute> { <constant> }
```

For example, the following production rule applying this formalism concerns a certain liver disease.

```
if
   Same patient sex female and
   LessThan patient age 30 and
   Same patient cholestasis intrahepatic and
   Same biochemistry hypergammaglobulinemia yes and
then
   Assert patient diagnosis chronic-active-hepatitis with
   CF = 0.60
fi
```

In this rule there is an additional entity, "CF=0.60", delimited by the keyword "with", a measure of uncertainty that is used by the expert to express his lack of certainty of the conclusion based on the four specified conditions. This uncertainty measure will be discussed briefly in section 3.

The actions in a conclusion of a production rule are one of the major departures from predicate calculus. In most rule-based systems, several are available. For example in the DELFI- 2 system, there is in addition to the "Assert" action (called "Conclude" instead) also an "Execute" action (used for invoking a procedure denoted by an attribute) and a "Write" action. The latter is a bit anomalous, because it has a textual message as its argument, instead of an object and attribute. This is not conform our simplified syntactical definition. In other systems, for example the OPS5 system there are additional actions, such as the "Modify" action, that replaces the former value of an attribute with the current value, specified in the production rule. This seems a suitable action for simulating process control. Deleting facts under certain circumstances by a "Delete" action may also be useful.

2.3. Data structures and knowledge representation related

We have discussed some methods for representing knowledge by means of symbolic representation schemes. Often, here a discussion on knowledge representation ends, leaving issues of implementation to the reader. However, we think that implementational issues are worth to be treated, hence this subsection about data structures in rule-based expert systems. We use a Pascal-like formalism for describing data structures. A LISP-like specification, could equally well be used, for the COMMON LISP

procedure "defstruct" is similar to the Pascal record type.

First, a production rule can be specified by means of record data types and pointer types, and it seems advantageous to use a description that is very similar to the knowledge representation formalism. If we place all production rules into a linked list, the following data structure is adequate:

```
rulePointer = pointer to ruleNode;

ruleNode = record
             rulenumber : cardinal;
             used       : boolean;
             antecedent : conditionPointer;
             consequent : conclusionPointer;
             next       : rulePointer
           end
```

The field *used* is set to **true** if the rule has been applied once, successfully or not. This helps in preventing circular reasoning, in which a production rule is used more than once. The conditions of each rule are also represented as a linked list:

```
conditionPointer = pointer to conditionNode;

conditionNode = record
                  predicate : pointer to symbolTable;
                  object    : objectPointer;
                  attribute : attributePointer;
                  values    : valuePointer;
                  andLink,
                  orLink    : conditionPointer
                end
```

The fields *andLink* and *orLink* represent the logical operators **and** and **or**. The data structure for conclusions is similar.

The data structure used for representing an attribute is also rather straight-forward, using a record data type with a number of fields that represent the various characteristics of an attribute:

```
attributePointer = pointer to attributeNode;

traceClass = (goal, askfirst);

attributeNode = record
                  name,
                  translation,
                  prompt       : string;
                  traced       : boolean;
                  class        : traceClass;
                  constraints,
                  values       : valuePointer;
                  next         : attributePointer
                end
```

If an attribute has the trace class *goal* it is considered to be a goal in the inference process. As such it triggers rule selection and application, issues to be discussed in the next section. An *askfirst* typed attribute, is asked before attempting to derive its value from production rules. If all possible values of an attribute have been determined, the field "traced" is set to **true**.

We conclude that there is an obvious analogy between our developed data structures and facilities for knowledge representation, even in a Pascal-like language.

3. Fundamental Operations In Expert Systems

Inference techniques and knowledge representation are strongly interrelated. Various search processes take part in the inference process, particularly in the selection and evaluation of knowledge items, for example production rules. Generally, two basic inference methods can be distinguished [5]:

- In top-down inference, starting with goal attributes, production rules are selected on the basis of a partial or total match with some pattern in the conclusion of the rules, and subgoals are generates as a result of production rule evaluation;
- In bottom-up inference, production rules are only applied if enough facts have been accumulated to evaluate the conditions. This process continues as long as new facts can be produced and thus additional rules can be applied.

3.1. Top-down inference

We start with a discussion of top-down inference on a rule base. In its simplest form, top-down evaluation start with a set of goals $\{G_1,G_2,...,G_n\}$ $(n \geq 0)$ and tries to proof each of these, applying the production rules in the rule base. The overall purpose of the process is to generate a set of facts that does or does not confirm goals and subgoals generated by the inference engine. Production rules are selected on basis of a partial or a total correspondence between a goal and one of the conclusions. In that way more than one production rules may be selected in order to confirm a goal. This selected set of production rules $\{R_1,R_2,...,R_m\}$ $(m \geq 0)$ is called the conflict set, because it is quite often not clear at all which of the rules should be processed first. Furthermore, it is also possible that none of the rules is applicable to the derivation of a goal or subgoal. In most systems this leads to questioning the user to enter one of the possible values. This so-called tracing process, is done in the following way:

```
procedure Trace(goal)

   Infer(goal);
   if [not traced] then
      Ask(goal)
   end
end
```

A goal might be a structured object, in which case it is an object with an associated attribute: Trace(object, attribute).

In the latter case, production rules are selected on the basis of a partial match operation in the procedure Infer described below, because in most expert system shells, and also in the DELFI-2 system, all rules that might contribute values to an object's attribute are collected.

```
procedure Infer(object, attribute)

   SelectRules(rulebase, object, attribute, selectedRule);
   while selectedRule ≠ nil do
      Apply(selectedRule);
      if [attribute traced] then
         return
      end;
      selectedRule := Next(selectedRule)
   end
end
```

The procedure Apply evaluates both conditions and conclusions of a production rule, in many cases creating subgoals, that are traced in a way similar to goals.

```
procedure Apply(selectedRule)

   EvaluateConditions(selectedRule);
   if [not failed] then
      EvaluateConclusions(selectedRule)
   end
end
```

A production rule fails if at least one of the conditions nested within and-operators, or all the conditions nested within or-operators, are evaluated to be false. Otherwise, a rule is said to succeed. The conclusions of a rule are only processed if the rule succeeded. The SelectRules algorithm processes the rule base as follows:

```
procedure SelectRules(rulebase, object, attribute, selectedRule)

   selectedRule := nil;
   rule := rulebase;
   while rule ≠ nil do
      if Match(object, attribute, rule.conclusion) then
         New(ruleSelected);
         ruleSelected.rule := rule;
         Next(ruleSelected) := selectedRule;
         selectedRule := ruleSelected
      end;
      rule := Next(rule)
   end
end
```

In the simple knowledge representation scheme for production rules, in which only propositions are available, the Match operator returns the value **true** only when the goal proposition and conclusion are identical. In the more sophisticated knowledge representation scheme, using objects and attributes, a goal and a conclusion match if object and attribute in both conclusion and goal or subgoal match. The specified values in the rule are ignored. In object-attribute-value representation, predicates operate on the collected facts and their arguments, by an attached function. Thus, arguments are actually evaluated.

A facility that is often added to the inference engine after the initial design has been completed, is a so-called look-ahead facility. This facility is similar to the rule application algorithm, but rule conditions are only checked on truth value, and are not evaluated. Thus, attributes are not traced but only scanned. For example, it could be the case that the third condition of a production rule, having four conditions nested in and-operators, is known to be false beforehand. In this case, there is no point in evaluating the first two conditions, because it is already known that this rule cannot succeed. However, the rule evaluation algorithm is not capable in detecting this, but the look-ahead facility is. Look-ahead leads to a remarkable pruning of the search space, and prevents many irrelevant questions to the user.

As mentioned before, on many occasions more than one rule is selected. Most rule-based systems with top-down inference, use backward chaining as a solution to the problem of the non-deterministic application of production rules and evaluation of the conditions of each selected rule. Usually, scheduling is done on a first-in-first-out basis. Once selected, production rules are applied sequentially, which is a rather trivial solution to the problem of conflict resolution. More sophisticated scheduling of rule subset evaluation involves a conflict resolution strategy that operates by means of priority scheduling. Algorithms have been developed to assign priorities to production rules, based on criteria such as the number of conditions in a rule or the measure of uncertainty of the conclusion. The priority criteria are often chosen dependent on available domain-specific control knowledge, i.e. information that is part of the solution process in a certain domain. More general schemes use potential path lengths, the degree of nodes in the corresponding AND/OR graph or some other measure of cost. However, these general schemes are only useful when no heuristic control knowledge is available, otherwise the conflict resolution strategy tends to degradate. Heuristic control knowledge is often only implicitly present in the knowledge base in the way it has been structured. Hence, the overall structure of the knowledge base is task dependent.

To illustrate the foregoing let us consider a simple rule base, in which only attributes have been specified.

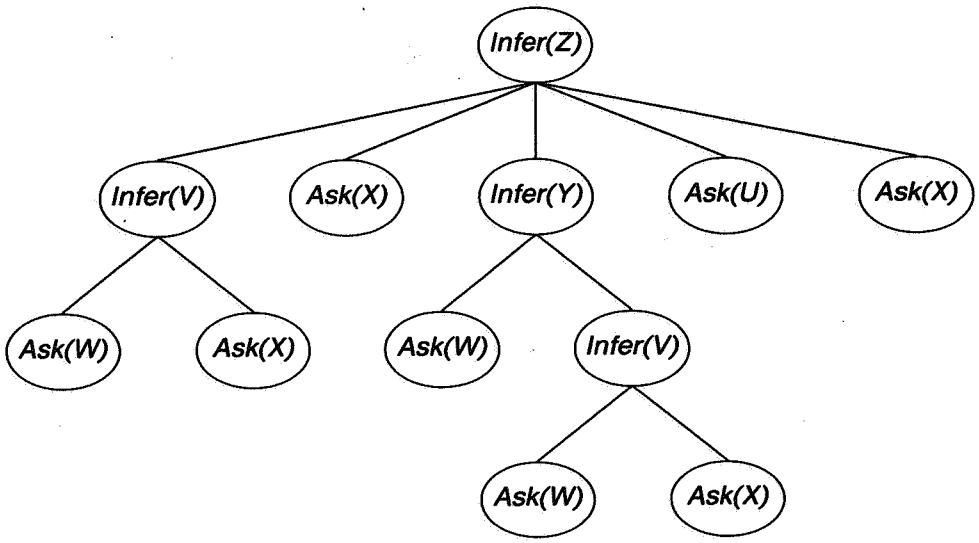

Figure 2. Search space tree for backward chaining

```
rule 1: if W, X then V fi
rule 2: if W, V then Y fi
rule 3: if V, X, Y then Z fi
rule 4: if U, X then Z fi
```

Starting with "Z" as a goal attribute, the system first selects rules 3 and 4 in the conflict set. When rule 3 is evaluated, the unknown attribute "V" will be met. Since rule 1 concludes about this attribute, this is the next selected rule. The tracing process is performed recursively. In this simple rule base, the attributes "W" and "X" or "W", "X" and "U" are finally asked of the user because there are no matching conclusions in any rule. In figure 2 the search space of the problem is specified. Top-down inference is often implemented as a form of depth-first search, i.e. pre-order traversal of a search tree, so the tree is traversed from top to bottom and from left to right. The search space tree can be compressed by deleting redundant nodes. This is a consequence of the set of facts, discussed earlier, which results in tracing a attribute not more than once.

A search space graph is depicted in figure 3. If an attribute already has been traced by the inference engine, it is considered superfluous to revisit such a node in the tree. A more complicated situation arises when a rule-based representation scheme is extended by an object-centered representation scheme. This extension provides the system with additional flexibility, because production rules are organized in objects, and rule invocation can only take place when the corresponding object in the conclusion of the rule is instantiated. Thus, an object-centered approach produces a flexible modular system. This has additional advantages from an implementational point of view. When selecting applicable production rules it is not necessary to scan the whole rule base. Thus, the inference process starts in another way then described until now; it starts with the instantiation of an object and traces the attributes that are the actual goals.

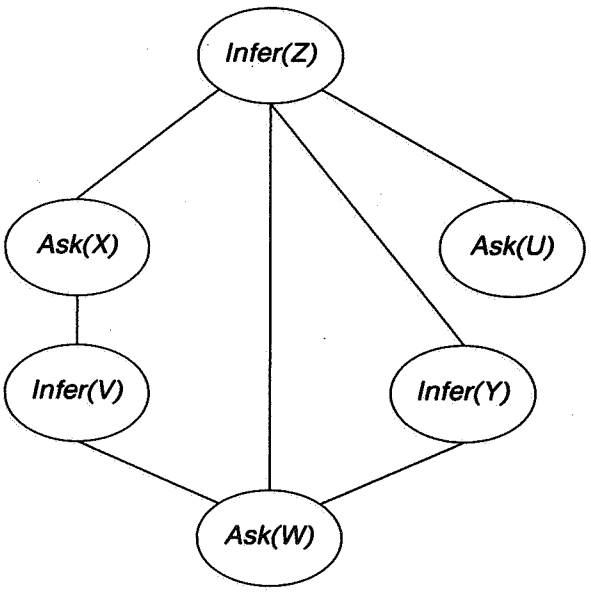

Figure 3. Search space graph

```
procedure Instantiate(object, template)

   New(object);
   CopyAttributes(template, object);
   attribute := First(object.attribute);
   while attribute ≠ nil do
      if [goal attribute] then
         Trace(object, attribute)
      end;
      attribute := Next(attribute)
   end
end
```

Facts are collected within the objects that have been dynamically instantiated, which simplifies easy retrieval.

This object-centered approach in a rule-based system is the first step towards a combined frame-based and rule-based system. Frames are objects with additional information, in particular information that concerns the mutual relationship between objects. Already mentioned is the ISA-link that structures a set of objects into a hierarchical net with property inheritance as a built-in inference method. Because of the specialization and generalization organization imposed on the objects, certain objects might inherit information of more general objects. This circumvents rule application and also prevents questions to the user of those values that can just as well be inferred from other objects.

3.2. Bottom-up inference as control

As stated before, bottom-up inference starts with data and applies production rules until a termination criterion is fulfilled. Bottom-up inference is often applied in domains where a lot of data must be processed, thus in essentially data-driven applications. In situations where an expert system is dedicated to consultation purposes, it seems more appropriate to use top-down inference, because this provides the user with a more consistent behaviour of the system. However, in an expert system building tool provided with top-down inference, there are many circumstances in which bottom-up inference could supplement the control strategy. This will be the next subject in our discussion.

In the first place, production rules are sometimes used within an expert system, not as part of the overall derivation of attribute values, but instead to perform certain actions if all of the conditions are fulfilled. This is, for example, the case in the following rule:

```
if
   GreaterThan engine temperature 150
then
   Execute engine overheat-switch(temperature)
fi
```

This rule succeeds if the temperature within an engine is higher than 150 centigrade. It switches on the overheat indicator, and shows the relevant temperature on the operator panel. This rule can never be part of the top-down inference process, because it is not intended to find any values of an attribute (nevertheless, this attribute might get assigned a value within the invoked procedure).

Another example of a rule that is never used by top-down inference, might be one that contains an object but neither an attribute nor a value. The purpose of such a rule might be to instantiate the specified object conditionally and trace grouped attributes only after this instantiation has taken place. An example of such a rule might look like this:

```
if
   Same aircraft pressurization-failure yes
then
   Instantiate pressurization-test
fi
```

This rule states that the object "pressurization-test" should be instantiated (including its attributes) when a pressurization failure occurred in an aircraft.

Still another application of bottom-up inference is to influence the top-down inference process itself. In many domains it is not definitely known beforehand which of the attributes of which objects are taken as a goal for the consultation. The selection of goals might depend of certain input data. Thus, a system that starts with bottom-up inference on initial data, generates goals by applying production rules, which are passed to the top-down inference engine. From there on, the system proceeds as usual in top-down inference.

The following example rule illustrates such a situation:

```
if
  Same patient disease malignant
then
  SetTrace patient prognosis goal
fi
```

The "SetTrace" action changes the trace class of an attribute "prognosis" to the goal state. Rules that influence the control strategy of the inference engine are often called meta-rules.

3.3. Approximate reasoning

In the foregoing, methods for knowledge representation and inference processes have been discussed. These techniques are based on methods derived from symbolic logic. Although, in essence, inference is based on symbolic computation, some methods for inexact or plausible reasoning take frequently part in the inference process. Logic has not enough expressive power to model real-world problems in an expert system, because of the occurrence of incomplete and inexact data in almost every domain. The approach of many researchers in the field has been to supplement logic with methods related to probability theory. For example, in 1975 Shortliffe and Buchanan proposed a method for the expression of judgemental belief [1]. This CF (certainty factor) model has been incorporated in MYCIN, and later in slightly modified form, in EMYCIN. It has also been used within the DELFI-2 system. Other well-known techniques include the Dempster-Shafer theory of evidence and the possibility theory of Zadeh [6,7].

In expert systems, two levels where certainty factors are employed, can be distinguished:
- the level of the expert, who wishes to express his uncertainty about his judgements on any conclusion in a rule;
- the level of the user, who may be doubtful about the accuracy of entered data.

There are various moments in the evaluation of a production rule where a measure of uncertainty should be taken into account. The evaluation of a condition needs an extension of the predicate function, in which case a predicate not only returns the value **true** or **false**, but in addition some measure of uncertainty, such as a certain factor. The situation is even more complicated, because in the EMYCIN system for example, the value **true** is only returned if the certainty factor exceeds some threshold value. In most formalisms, conjunctions and disjunctions of conditions effect the resulting certainty factor. If conditions form a conjunctive expression, then:

$$CF = min\{CFcondition_1,...,CFcondition_n\}$$

In the case of a disjunction of conditions we get instead:

$$CF = max\{CFcondition_1,...,CFcondition_n\}$$

One well-known problem in modelling uncertainty in expert systems is how to combine various certainties related to an attribute value into one certainty factor. The formula used in the EMYCIN system has been successfully applied in various applications, in spite of theoretical objections. Sensitivity experiments with MYCIN revealed the used CF model to be quite stable to variations. However, this phenomenon may be due to the short inference chains in MYCIN.

In an example the application of the CF model will be illustrated. There are only two rules in the knowledge base present:

rule 1: **if** *B, C* **then** $A_{CF=0.4}$ **fi**
rule 2: **if** *D* **then** $A_{CF=0.3}$ **fi**

It is assumed that the attributes "B", "C" and "D" are traced with certainty factors 0.5, 1 and 1 respectively. The following two computations are performed:

1. The minimum of the CF's in the conditions is determined and multiplied by the certainty factor in the conclusion:

$$CF = CF_{conclusion} \cdot min\{CFcondition_1, CFcondition_2, ...\}$$

   In this case the following sequence of computations is performed: rule 1 results in: 0.4·min{0.5, 1} = 0.2, and rule 2 in: 0.3. So fact "A" has two certainty factors generated by different sources.
2. In the next stage these certainty factors are combined by applying the combination rule of Shortliffe and Buchanan that in this case results in: 0.2·(1 - 0.2) = 0.44.

So the combination of certainty factors increases the evidence about the values of the attributes, but not by simply adding the two factors.

4. A Medical Application: HEPAR

One of the expert systems being developed using the DELFI-2 system, is a medical system for the diagnosis of liver disease. Medicine has traditionally been a rich field for Artificial Intelligence and has resulted into a large variety of experimental systems [8]. Medicine is a suitable test area for expert systems, because it offers many interesting problems with completely different problem and solution characteristics. In addition, medical applications appeal many researchers for their potential beneficial effects on the health service.

The area of liver disease is a field that is far from being formalized: the medical knowledge is characterized by its incomplete, inexact and symbolic nature.

In co-operation with Dr. A.R. Janssens of the Department of Gastro-enterology of the University of Leyden work is being done on the construction of an expert system that deals with liver disease. As an intermediate goal, the effort is directed at the design of a system that is able to produce a medical diagnosis on the basis of entered patient findings and certain laboratory data. In the future we probably will try to include treatment and prognostic considerations into the system. This expert system, called HEPAR, incorporates both clinical experience and data from medical literature. The organization of the knowledge base reflects the usual diagnostic approach in the area of liver disease.

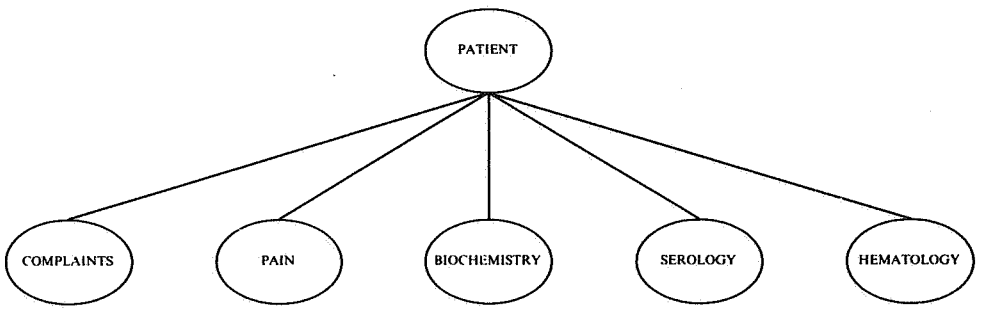

Figure 4. Structuring diagnosis in liver disease

The objectives of the research are:
- to improve diagnostic capabilities of physicians;
- to make useful medical knowledge easily available to non-experts.

The expert system HEPAR needs the following simple data for proper functioning:
- general patient data, such as age and sex;
- the medical history of the patient;
- data obtained from the physical examination;
- laboratory data and serological findings;
- ultrasound and X-ray findings.

The system proceeds from the outset to the following intermediate conclusions:
- the type of cholestasis (bile congestion): intra- or extrahepatic;
- the presence of liver failure;
- malignant or benign liver disorder.

Together with other characteristic patient data a diagnostic conclusion is reached.

The following rule which is part of the diagnostic expert system HEPAR, concludes about a possible disease state.

```
rule 115
If
  1.0 the duration of complaints, clinical signs or lab
      abnormalities is chronic
  2.0 the sex of the patient is female
  3.0 the complaint of the patient is Raynaud's_phenomenon, or
  3.1 the complaint of the patient contains burning_eyes and
      dry_mouth
  4.0 the cholestasis of the patient is extrahepatic
then
  1.0 there is suggestive evidence (0.60) that the possible
      diagnosis of the patient is primary_biliary_cirrhosis
```

Another part of the knowledge base contains a structured overview of the problem domain which is stored as objects in a tree-like structure, that is depicted in figure 4.

The system actually uses very few laboratory data in spite of its specialized nature. In addition, invasive techniques, such as ERCP and liver biopsy take no part in the diagnostic process of the system. Thus, maximum use is made of easily gathered clinical data.

At present, the expert system HEPAR is being tested, using patient data both from the University Hospital of Leyden and Rotterdam.

ACKNOWLEDGEMENT

I want to express my gratitude to Linda van der Gaag for commenting on an early version of this paper and to Henk de Swaan Arons for the pleasant co-operation in the DELFI-project. I also like to thank Roel Janssens for spending so much time in building the expert system HEPAR.

5. REFERENCES

[1] BUCHANAN B.G., SHORTLIFFE E.H. (1984). *Rule-based expert systems: the MYCIN experiments of the Stanford Heuristic Programming Project.* Addison-Wesley, Reading, Massachusetts.

[2] DE SWAAN ARONS H, LUCAS P.J.F. (1984). Expert systems in an application oriented environment. *Informatie,* Volume 26, 8: 631-637 (in Dutch).

[3] HAYES-ROTH F., WATERMAN D.A., LENAT D.B. (1983). *Building expert systems.* Addison-Wesley, Reading, Massachusetts.

[4] NILSSON N. (1980). *Principles of artificial intelligence.* Springer-Verlag.

[5] KOWALSKI R. (1979). *Logic for problem solving.* North-Holland, New-York.

[6] PRADE H. (1985). A computational approach to approximate and plausible reasoning with applications to expert systems. *PAMI,* Volume 7,3: 260-283.

[7] GUPTA M.M., KANDEL A., BANDLER W., KISZKA J.B., EDITORS. (1985). *Approximate reasoning in expert systems.* North-Holland, Amsterdam.

[8] LUCAS P.J.F., JANSSENS A.R. (1985). Medical expert systems: an aid in diagnostic and therapeutic decision-making. *Nederlands Tijdschrift voor Geneeskunde,* Volume 129, 4: 160-165 (in Dutch).