Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

A. Eliëns

Expert systems as deductive systems

Computer Science/Department of Software Technology Report CS-R8825 July

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.).

Expert Systems as Deductive Systems

A. Eliëns
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

This paper characterizes rule-based expert systems as deductive systems. Deduction is viewed as an operator transforming a given state of knowledge into a new state containing as least as much knowledge. The meaning of a knowledge base can then be defined as the deductive closure obtained by applying the rules to a set of initial facts. An interpreter is defined that models the operational aspects of the inference engine of Mycin-like expert systems. The interpreter is proven complete provided that the dependencies between the attributes occurring in the rules are acyclic. First, the knowledge representation language will be restricted in such a way that the reasoning capabilities of the system do not transcend a positive logic. Later the knowledge representation language will be extended to adequately capture the functionality of existent rule-based expert systems. To model these extensions a four-valued logic is used, similar to the logic of Belnap [Be77]. This work can be seen as a case study in the semantics of rule-based expert systems.

Keywords & Phrases: Expert Systems, Semantics

1982 CR Categories: I.2.1 (Artificial Intelligence: Applications and Expert Systems),
F.3.2 (Logics and Meanings of Programs: Semantics)

1980 Mathematics Subject Classification: 68G15 (Theorem proving), 68B10 (Semantics)

The work in this document was conducted as part of the PRISMA project, a joint effort with Philips Research Eindhoven, partially supported by the Dutch "Stimulerings-projectteam Informatica-onderzoek" (SPIN).

Report CS-R8825
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

1. Introduction

The notion of deductive systems was introduced in [Ta56] and exploited in [Cl74] to describe formal systems extending propositional logic and first order predicate calculus. In this paper work is described which aims at a semantical characterization of rule-based expert systems along these lines, using techniques from denotational semantics.

This paper gives a detailed description of a simple kind of expert systems, covering both declarative and procedural aspects:

- A *knowledge base* is described with respect to its syntax and semantics. The notion of facts and rules over a domain of discourse is introduced. A natural interpretation of rules as operators on a set of facts is presented, and the meaning of the knowledge base is defined with respect to this interpretation.
- An *inference engine* is presented that, conjoined with a knowledge base, produces inferences over the domain of discourse. The characterization given deals also with the more operational aspects of expert systems, notably with the strategy of inference, and the possibility to dynamically add facts.
- Some extensions with respect to the knowledge representation language are discussed and it is indicated how to adapt the framework to cope with these extensions.

In this paper, I've chosen to closely follow the syntax and semantics of rule-based expert systems as presented in e.g. [BS84] and [Lu86]. Such a choice is debatable. Cf. [Be87]. Nevertheless, despite justifiable criticism on this particular formalism, it has been the primary vehicle for representing knowledge in many existing (diagnostic) expert systems [Bu83]. A disadvantage of this approach is that it seems to necessitate a (notationally) rather complex treatment, due, at least partly, to the more operational than declarative nature of the formalism. In the light of these remarks, this work can be seen as a case study in the semantics of rule-based expert systems. For an introduction to the subject, see [BS84] and [Lu86].

1.1. Rule-based expert systems

Rule-based expert systems typically consist of a knowledge base containing facts and rules, and an inference engine that is used to derive facts. Facts can be considered as items that affirm that a particular attribute has a particular value or that deny that such is the case. With respect to the rules contained in the knowledge base two distinct points of view are possible. One can regard the rules as implications in the logical sense and take the rules and facts together as a collection of axioms describing some particular state of the world. Or one can regard only the facts initially given to the system as axioms and the set of rules, that in some sense embody the knowledge, as the rules of inference that enable a transition from some state of knowledge to another state. The former view has the advantage that the meaning of the knowledge base can be stated in a way independent of the inference engine, namely as a model satisfying the theory corresponding to the set of axioms. Another advantage is that one needs only a single inference rule, say (linear) resolution, to explain the (desired) working of the inference engine. An advantage of the latter viewpoint however is that it corresponds more closely with the actual (sometimes ill-defined) nature of expert systems, which in most cases can be considered as an outgrowth of production-systems. To give an example, in MYCIN [BS84] the rules contain explicit instructions to test a condition and explicit instructions to perform some action. As will be shown later, for a very restricted repertoire of instructions actually mentioning these instructions is purely redundant. However this does not hold in general.

1.2. Deductive systems

When regarding a knowledge base as a set of facts and a set of mappings that enable one to generate facts from facts the meaning of an expert system must be described in terms of states of knowledge that are attainable by applying the rules.
Taking this latter viewpoint endangers the project of giving sensible meaning to a knowledge base unless we impose some restrictions on the kind of mappings that are involved. As one desideratum it can be stated that the mapping we like to associate with the collection of rules is *conservative* in the sense that it does not destroy information that is contained in the state upon which it operates. Moreover we would like that some unique meaning, some final state of knowledge, can be obtained after which applying the rules does not add any information.

DEFINITION 1. 2: A *deductive system* is defined as a system consisting of a set of sentences S and a consequence mapping C such that:

(*i*) S is denumerable

(*ii*) $X \subseteq S$ implies $X \subseteq C(X) \subseteq S$

(*iii*) $X \subseteq S$ implies $C(C(X)) = C(X)$

(*iv*) $X \subseteq S$ implies $C(X) = \cup\{ C(Y) \mid Y \subseteq X,\ Y \quad \text{finite} \}$ □

REMARKS: Condition (ii) states that the consequence mapping must be conservative in the sense that it does not destroy information.
Condition (iii) requires that the consequence mapping is idempotent, which means that the mapping extracts all information from the given set of sentences.
Condition (iv) states a compactness condition, requiring that the consequences of an infinite set of sentences can be obtained as the union of the consequences of finite subsets of the given set.
If condition (iv) does not hold but the condition

(*iv'*) $X \subseteq S$ implies $C(X) \subseteq \cup\{ C(Y) \mid Y \subseteq X, Y \quad \text{finite} \}$

does hold, then the system will be called a pseudo-deductive system.

Note that condition (*iv*) implies that the mapping C is monotonic.

A deductive system, in our sense, then can be defined as a mapping C from a set of propositions to a set of propositions that can be characterized by the requirements that it is *conservative*, and preferably *monotonic*. For comparison, a similar approach to deductive reasoning is followed in [Sa85] and to some extent in [LM83]. See also [Ma87].

To summarize the contents of this paper, for the simple system, reasoning with positive logic, the meaning of the rules can be presented as the least fixed point of an operator on states (of knowledge). The ordering on states is that induced by set inclusion. A state is considered to contain more information if it contains more facts. The meaning obtained in this way is compared with the derivations produced by an interpreter that models the inference engine. This interpreter is proven complete with respect to the intended meaning of the knowledge base, provided that the rules contain no direct or indirect recursion on an attribute. Curiously enough, however, the format of the rules allows to overcome this restriction by some special tricks.
For the extensions to the knowledge representation language it seems harder to give a characterization of the intended meaning. To model some of these extensions a four-valued logic is used, similar to the logic presented by Belnap [Be77]. Such a logic has also been used by Sandewall [Sa85] to deal with non-monotonic reasoning. This subject will however not be further explored in this paper. To stress the difference between declarative and procedural semantics it is remarked that the interpreter

requires only minor adaptations to adequately model the operational meaning of these extensions. Cf. [JM84].

2. The knowledge base

A knowledge base consists of facts and rules about some domain of discourse. The meaning of a knowledge base can, roughly, be characterized as the set of facts that can be derived by applying the rules.
Deduction is viewed as an operation on a set of facts. The meaning of a rule is expressed as an operator mapping a set of facts into a set of facts containing the original facts. Whenever the mapping corresponding to the rules is not conservative with respect to its input then it might be possible that no state containing all the information implicit in the initial facts and rules will be attained. An example will be given that illustrates this situation.

2.1. Syntax

In this section we will restrict ourselves to the description of a system that reasons with 'positive logic' only. We closely follow the syntax as described in [BS84].

Definition 2. 2: *knowledge representation*
1. Triples of the form $\langle o,a,v \rangle$ are called items. Informally they can be regarded as propositions stating that the attribute a of object o has value v. Objects, attributes and values are drawn from respectively the finite sets O, A and V. The set $D = O \times A \times V$ is called the domain of discourse. Items, belonging to D, are indicated by d.
2. Rules are of the form $c_1,...,c_n \rightarrow ac_1,...,ac_m$
with c_i, $i = 1,...,n$, *conditions* of the form *pred d* with *pred* a predicate symbol from a finite set *Pred*, and with ac_i, $i = 1,...,m$, *actions* of the form *act d* with *act* an action symbol from a finite set *Act*.
In the sequel it will be assumed that $Pred = \{ same \}$ and $Act = \{ add \}$ unless explicitly stated otherwise.
3. Assume the sets *Pred* and *Act* to be fixed. A knowledge base $KBS_D = (It,R)$ is defined to consist of a designated set of items $It \subseteq D$, called initial facts, and a set of rules R, with conditions and actions containing predicate and action symbols from respectively *Pred* and *Act*.

Remarks:
- When the context allows the subscript D in KBS_D may be omitted.
- A condition can be regarded as an instruction to perform a test with regard to the specified item, or put differently, to establish the propositional truth of an item. An action is to be considered as an instruction to assert the truth or falsity of an item.
- Under the restriction that $Pred = \{ same \}$ and $Act = \{ add \}$ it is evident that rules are closely related to ground Horn sentences. Let π be the projection of conditions and actions to their item part. Transform $c_1,...,c_n \rightarrow ac_1,...,ac_m$ to $\{ (\pi(c_1),...,\pi(c_n) \rightarrow \pi(ac_1)) \ ,..., \ (\pi(c_1),...,\pi(c_n) \rightarrow \pi(ac_m)) \}$ and take the union of the transformations of the rules. Later an example will be given, however, where the syntactic format actually makes a difference, under a particular strategy of inference.

2.2. Semantics

Deduction will be regarded as a mapping that transforms a state of knowledge. For reasons of generality, states are represented as functions: the characteristic functions of sets of propositions. Obviously, with each state corresponds a set of propositions.

DEFINITION 2. 3: *state*
Let

$$\chi(S)(d) = \text{ if } d \in S \text{ then } \mathit{true} \text{ else } \mathit{false} \text{ fi}$$

Let $\Sigma^* = \{ \Sigma \mid \Sigma = \chi(S), S \subseteq D \}$ be the set of states. Σ^* coincides with the characteristic functions of subsets of D.
For each state Σ the set associated with Σ is

$$|\Sigma| =_{df} \{ d \mid \Sigma(d) = \mathit{true}, d \in D \}$$

□

The evaluation of conditions and actions occurs in a state. The evaluation of a condition delivers either *true* or *false*. The evaluation of an action in a particular state possibly results in a modification of the input-state.

DEFINITION 2. 4: *evaluation*

To evaluate conditions and actions we need the valuations γ and α defined, for $Pred = \{ same \}$ and $Act = \{ add \}$ by

$$\gamma(same\ d)(\Sigma) = \Sigma(d)$$
$$\alpha(add\ d)(\Sigma) = \chi(|\Sigma| \cup \{ d \})$$

□

REMARK: The restriction to $Pred = \{ same \}$ and $Act = \{ add \}$ is deliberate. See also the remark following the definition of the meaning of the knowledge base.

LEMMA 2. 5:

Let the ordering on truth values be such that $false < true$ and the ordering on states be the ordering induced by set inclusion, then for $Pred = \{ same \}$ and $Act = \{ add \}$

(*i*) γ is monotonic in the state-argument, and

(*ii*) α is monotonic and conservative in the state-argument.

PROOF: Note that the notion *conservative* amounts to requiring that for $|\Sigma| \subseteq |\alpha(ac)(\Sigma)|$ for arbitrary Σ and ac. □

Each rule corresponds to a state transforming mapping. Given a set of rules R, the state-transformer corresponding to R is defined to be the union of the mappings corresponding to the rules in R. The effect of applying R is that all actions are executed occurring in the rules for which the conditions evaluate to *true*.

DEFINITION 2. 6: *deduction*

Given a set of rules R, with $R \neq \emptyset$, and state function Σ
let $[\![\cdot]\!] : \Sigma^* \rightarrow \Sigma^*$, mapping state functions to state functions, be defined by

$$[\![R]\!] (\Sigma) = \bigcup_{i=1}^{k} [\![r_i]\!] (\Sigma) \quad \text{for } R = \{ r_1, \ldots, r_k \}$$

with

$$[\![c_1,\ldots,c_n \rightarrow ac_1,\ldots,ac_m]\!](\Sigma) = \begin{cases} \alpha(ac_m)(\cdots(\alpha(ac_1)(\Sigma))) & \text{if } \forall i (1 \leq i \leq n)\ \gamma(c_i)(\Sigma) \\ \Sigma & \text{otherwise} \end{cases}$$

where

$$\bigcup_{i=1}^{k} \Sigma_i =_{df} \chi(\bigcup_{i=1}^{k} |\Sigma_i|)$$

□

LEMMA 2. 7: $[\![R]\!]$ is monotonic and conservative.

PROOF: Omitted. □

To obtain the meaning of a set of rules, we have to define the closure of a state under a set of rules.

DEFINITION 2. 8: *closure*

Let for arbitrary Σ

$$[\![R]\!]^0(\Sigma) = id$$
$$[\![R]\!]^{i+1}(\Sigma) = [\![R]\!]([\![R]\!]^i(\Sigma))$$

and define the meaning of $KBS_D = (It,R)$ by

$$[\![KBS_D]\!] =_{df} \bigcup_{i=0}^{\infty} [\![R]\!]^i(\chi(It))$$

□

Next, in close analogy to the fixed-point semantics for sets of Horn clauses given in [LM83], the meaning of a knowledge base can be defined as the set of facts generated by the union of all possible sequences of rule applications.

THEOREM 2. 9: *meaning*

The meaning of $KBS_D = (It,R)$ is given by

$$[\![KBS_D]\!] = [\![R]\!]^k(\chi(It)) \text{ for some finite } k$$

PROOF: The least fixed point of the operator $\lambda\Sigma.[\![R]\!](\chi(It) \cup \Sigma)$ can be obtained by $\cup_{i=0}^{\infty} [\![R]\!]^i(\chi(It))$, where $[\![R]\!]^0$ is taken to be the identity. Since $[\![R]\!]$ is conservative this is equivalent to $[\![R]\!]^{\infty}(\chi(It))$. From the assumption that the domains are finite it follows that a finite number of iterations suffices. □

As an alternative proof: For $Pred = \{ same \}$ and $Act = \{ add \}$ transform the rules to Horn sentences by the transformation described above and consult [LM83].

REMARKS:

· In order that a unique (reliable) meaning exists it has to be required that α is monotonic and conservative, in the sense that it preserves the information that is contained in the state-argument.

Consider for example the knowledge base with

$$It = \{ \langle o,a,v \rangle \}$$

$$R = \{ (same\ \langle o,a,v\rangle \rightarrow delete\ \langle o',a',v'\rangle, add\ \langle o'',a'',v''\rangle),$$
$$(same\ \langle o',a',v'\rangle \rightarrow delete\ \langle o'',a'',v''\rangle, add\ \langle o,a,v\rangle),$$
$$(same\ \langle o'',a'',v''\rangle \rightarrow delete\ \langle o,a,v\rangle, add\ \langle o',a',v'\rangle) \}$$

and *delete* defined by $\alpha(delete\ d)\ (\Sigma) = \chi(|\Sigma| \setminus \{ d \})$. It is evident that this knowledge base has no stable interpretation.
If $Act = \{ add \}$ this restriction of information preservation is satisfied. A special case occurs when also γ is monotonic, as with $Pred = \{ same \}$. The existence of a unique minimal fixed point is then relatively easy to prove.

· It can be observed that the number of iterations does not exceed the size of the domain of discourse *D*.

The deductive system to be associated with the rules can be characterized by the operator $\lambda S \subseteq D. |[\![R]\!]^*(\chi(S))|$. This operator can be relativized to the set of initial facts *It* by defining it as $\lambda S \subseteq D. |[\![R]\!]^*(\chi(It \cup S))|$. Clearly this operator has all the characteristics that were required in the introduction.

The characterization of a knowledge base as a deductive system allows for an interesting generalization of the notion of knowledge base. An observation due to Belnap [Be77] is that any mapping from sets of propositions to sets of propositions "behaves like a rule" as long as it is *monotonic* and *conservative*. This justifies the replacement of parts of the knowledge base by special purpose decision procedures suitable for the domain of expertise, provided that the semantics in terms of information-content is not affected.

3. Proof trees

Traditionally, a rather mechanical metaphor has been used to describe the reasoning of an expert system. Of course this is not harmful as long as the 'engine' behaves in some sense 'logical'. We take the view that what an expert system ought to do is to construct some kind of proof that shows that some item is either *true* or *false* and by means of the proof also shows *why*.

To enhance the intuition for the interpreter that will be described in the next section, we will allow ourselves a slight digression on proof trees.

Proof trees are simple structures that represent the dependency relations between propositions. This dependency reflects the rules used in constructing the proof trees, in that a proposition that labels a non-leaf node of the tree follows from the propositions labeling the children of the node on behalf of some rule. The leaf nodes are the propositions that correspond with the given set of initial facts.

Definition 3. 2: *proof trees*

For $KBS = (It,R)$

(*i*) For each $d \in It$, *d* is a proof tree compatible with *KBS*

(*ii*) If $T_1,...,T_n$ are proof trees compatible with *KBS* and $d \in D$ then $d \leftarrow T_1,...,T_n$ is a proof tree compatible with *KBS*, with $root(d \leftarrow T_1,...,T_n) = d$ and subtrees $T_1,...,T_n$, if there is some rule $r \in R$ of the form $c_1,...,c_n \rightarrow ...,ac,...$ such that $\pi(c_i) = root(T_i)$, $i = 1,...,n$, and $\pi(ac) = d$.

□

Remarks:
· The subtrees T_i, $i = 1,...,n$, are considered to be unordered.
· Note that the latter construction works only for $Pred = \{ same \}$ and $Act = \{ add \}$

We will be interested in a special kind of proof trees.

DEFINITION 3. 3: A *regular* proof tree is a proof tree in which no proposition occurs more than once on a path from the root to one of the leaves.
□

Regular proof trees play an important role in that the top down interpreter, defined in the next section, will be restricted in such a way that it only constructs regular proof trees. The collection of regular proof trees compatible with a knowledge base will be characterized by describing a bottom-up procedure that is guaranteed to stop in a finite number of steps. To this end we need the following

DEFINITION 3. 4: *rule-base transformation*

Let again π be the projection of conditions and actions to their item-part. Say R^S is the result of *filtering R through S* for $S \subseteq D$ if R^S is obtained from R by removing every rule of the form $c_1,...,c_n \rightarrow ac$ if $\pi(ac) \in S$ and replacing every rule of the form $c_1,...,c_n \rightarrow ac_1,...,ac_m$ by a rule in which all ac_i, $1 \leqslant i \leqslant m$, are removed for which $\pi(ac_i) \in S$, and the entire rule is removed if no action remains. □

DEFINITION 3. 5: For a given set of initial facts It, T is called a proof tree w.r.t. S and R if T is a proof tree compatible with (It,R^S). □

Obviously T is a proof tree w.r.t. R and S, for $S = \emptyset$, if, by definition, T is an proof tree compatible with (It,R).

Now the procedure delivering all (regular) proof trees can be stated as follows:

DEFINITION 3. 6:

For $KBS_D = (It,R)$ define the sequence of pairs $(\overline{T}_i,R_i)$

$$\overline{T}_0 = It \ , \ R_0 = R^{It}$$

and

$$\overline{T}_{i+1} = \overline{T}_i \cup \{ d \leftarrow T_1,...,T_n \mid \exists r \equiv c_1,...,c_n \rightarrow ...,ac,... \text{ in } R_i \text{ with } d = \pi(ac) \text{ and } T_1,...,T_n \in \overline{T}_i \text{ and for } i = 1,...,n \ \pi(c_i) = root(T_i) \}$$

$$R_{i+1} = R_i^S \quad \text{for } S = \{ d \mid d = root(T) \text{ and } T \in \overline{T}_{i+1} \}$$

□

Now we can state

LEMMA 3. 7: Let $items(T)$ be the set of items occurring on the nodes of T and $items(\overline{T}) = \bigcup_{T \in \overline{T}} items(T)$. Then any $T \in \overline{T}_i$ is a proof tree w.r.t. $D - items(\overline{T}_i)$ and R, where D is the domain of discourse.

PROOF: Obvious. □

The construction amounts to building the set of regular proof trees in the sense that for no tree any item occurs more than once on any path in the tree.

The following theorem states that the construction is adequate.

THEOREM 3. 8: Let $KBS = (It,R)$, then $|[\![R]\!]^j(\chi(It))| = \{ root(T) \mid T \in \overline{T}_j \}$, for each $j \geqslant 0$.

PROOF: For $j = 0$ the result is immediate. Let the statement hold for $j = m$, and let $\chi_{It} = \chi(It)$. Assume $d \in |[\![R]\!]^{m+1}(\chi_{It})|$ then it has to be the case that for some rule $r \equiv c_1,...,c_n \rightarrow ...,ac,...$ $d = \pi(ac)$ and, for $i = 1,...,n$, $\pi(c_i) \in |[\![R]\!]^{m}(\chi_{It})|$. By assumption there are $T_i \in \overline{T}_m$ such that $\pi(c_i) = root(T_i)$, $i = 1,...,n$. Now suppose that r is not applicable, then by the construction of $\overline{T}_m$ d has already been derived. If this is not the case then d can be derived by applying r. The other part is trivial, since the construction described above imposes a restriction on all possible proof trees by the requirement of regularity. □

REMARK: A similar observation, that whenever there exists a proof tree then there also exists a regular proof tree, has been made in [LR81], in the context of establishing the relation between problem-reduction and resolution.

As will be clear from the discussion in the next section we can impose other constraints on the proof trees that are compatible with a knowledge base. This allows us to further sharpen our estimate of the cost of deriving the facts implied by a particular knowledge base.

Since in general we are not interested in all the information we can get, for reasons of efficiency a bottom-up construction is not acceptable. To prepare the way for the description of the interpreter, that will be given in the next section, we characterize, following [ABW86], a top-down interpreter constructing regular proof trees by defining a relation I_R as in

DEFINITION 3. 9: *top-down interpreter*

Let $KBS = (It,R)$ be a knowledge-base over D and let IT be the set of all proof trees. Then the relation $I_R \subseteq D \times IT \times 2^D$ is defined by

(*i*) $I_R(d,d,S)$ if $d \in It$ for all $S \subseteq D$

(*ii*) $I_R(d,d \leftarrow T_1,...,T_n,S)$ if $d \notin S$ and for some rule $r \equiv c_1,...,c_n \rightarrow ...,ac,...$ in R, with $\pi(ac) = d$, $I_R(\pi(c_1),T_1,S \cup \{ d \}),...,I_R(\pi(c_n),T_n,S \cup \{ d \})$

□

REMARK: The set S is like a state in which a record of the inference thus far is kept, and functions as a loop-trap to prevent unwanted recursion.

The following theorem states that a top-down interpreter can construct a regular proof tree if there exists one and that every proof tree in I_R is regular.

THEOREM 3. 10: $I_R(d,T,S)$ *iff* $d = root(T)$ and T is a proof tree w.r.t. S and R

PROOF: In order to render the intuition behind the proof it should be remarked that the notion compatible amounts to saying that the rules contain sufficient information for deriving the item considered. In the top-down interpreter each goal item is added to the set S embodying the loop-trap. The set used for filtering the rules, in the bottom-up construction of the proof trees, can, almost, be taken to be the complement of the loop-trap, with respect to the domain of discourse, in the sense that it grows along with the depth of the proof tree establishing the provability of an item. The proof proceeds by induction on the heigth of the proof tree.
($\Rightarrow$) For $T \equiv d$, $d \in It$, it immediately follows that T is compatible with (It,R).
For $T \equiv d \leftarrow T_1,...,T_n$ it holds that $d \notin S$ and for some rule $r \equiv c_1,...,c_n \rightarrow ...,ac,...$ with $d = \pi(ac)$ and $I_R(\pi(c_1),T_1,S \cup \{ d \}),...,I_R(\pi(c_n),T_n,S \cup \{ d \})$. By induction $\pi(c_i) = root(T_i)$ and T_i is a proof tree w.r.t. $S \cup \{ d \}$ and R. From this it follows that no T_i has d on its branches and since R^S allows d to be derived by r it follows that T is a proof tree w.r.t. R and S.
($\Leftarrow$) In accordance with the construction of regular proof trees, for $T \equiv d$ in $\overline{T}_0$ and T compatible with (It,R) it immediately follows that $I_R(d,d,S)$, for arbitrary S.

For $T \equiv d \leftarrow T_1,...,T_n$, with $T_1,...,T_n \in \bar{T}_k$, let T be compatible with R^S for some S. By the definition of compatibility $d \notin S$. By the requirement of regularity $T_1,...,T_n$ are regular proof trees compatible with $R^{S \cup \{d\}}$. By induction then $I_R(root(T_i),T_i,S \cup \{d\})$, $i = 1,...,n$. And hence $I_R(d,d \leftarrow T_1,...,T_n,S)$. □

COROLLARY 3. 11: For arbitrary S it holds that $I_R(d,T,S)$ *iff* $I_{R^S}(d,T,S)$

PROOF: From the previous lemma it follows that $I_R(d,T,S)$ implies T is a proof tree w.r.t. R and S, meaning that $d = root(T)$ can be derived by applying rules from R^S, the set of rules obtained by filtering R through S. Conversely, if T is a proof tree constructed by applying rules from R^S then, by the definition of R^S, no item can be both a non-leaf node in the tree and in S. □

REMARK: The top-down interpreter described prevents looping by simply keeping a set of goals on which no further expansion is allowed. In the context of actions modifying a state, marking rules that have been applied is allright whenever the action has become superfluous. In the expert system to be described, however, a less selective criterium both for the loop-trap is used, namely not the occurrence of an item of the form $\langle o,a,v \rangle$ but the occurrence of the $\langle o,a \rangle$ part. This has a drawback on the completeness of top-down inference unless certain restrictions are imposed on the rule-base.

4. THE INFERENCE ENGINE

The vital part of an expert system is the inference engine, that allows the user to query the system. When consulting an expert systems the user supplies the system with a set of initial facts and a query. The system then responds with an answer to that query, possibly after asking for some additional information.

4.1. Query evaluation

In this section we will concentrate in particular on the inference strategy employed by the (family of) expert system(s) in question.

The major issues concerning the inference strategy are rule selection and recursion control. The basic strategy followed by Mycin-like systems is a backward-chaining on goals. Rule selection, in the context of this strategy, involves finding rules with respect to some goal. Recursion control must me performed to detect cyclic dependencies and to prevent (infinite) looping. Both rule selection and recursion control are handled in a rather crude way in these systems. Instead of selecting (checking) on *⟨object,attribute,value⟩* triples, selection (control) only uses *⟨object,attribute⟩* pairs. This has severe implications for the completeness of the inference engine. Under certain restrictions, however, the interpreter can be proven complete. These restrictions can be circumvented, though, by means of some tricks, as will be illustrated in the remarks following the completeness theorem.

DEFINITION 4. 2: Some auxiliary domains and functions are needed

1. Let $p \in Pair \subseteq O \times A$ be a set of pairs, with the corresponding function π, defined by $\pi(\langle o,a,v \rangle) = \langle o,a \rangle$, the projection on the first two components. For convenience π will be used to extract a pair from either goals, conditions or actions.
2. Let $Status = \{ askable, derivable, derived \}$ and $\delta \in Pair \rightarrow Status$ be the function delivering the status of a pair. It is assumed that $\delta(p) = derivable$ for all p unless stated otherwise. The function δ will be used to control the inference.
3. A goal is defined to be of the form *same* d with $d \in D$. The set of goals coincides with the set of conditions. □

In addition to a knowledge base, an expert system is defined as a system consisting of a state, a set of rules, and a control part that determines the actual inferences, i.e. the transformations of the initial

state as evoked by the query from the user.

DEFINITION 4. 3: *expert system*

For a knowledge-base $KBS_D = (It,R)$ an expert system $ES_D = (\Sigma,R,\delta)$ over D is defined to consist of a state Σ, with $It \subseteq |\Sigma|$, a set of rules R and controle information δ. □

Starting the consultation the state Σ will be equal to $\chi(It)$, the initial state of *KBS*. During the consultation, when checking the conditions of the rules and executing the actions when appropriate, Σ will be modified to a state containing $\chi(It)$.

Let $\underline{ES}$ be a family of expert systems over D and let G the set of goals, then a query can be defined as a mapping

$$QE: \underline{ES} \times G \rightarrow \underline{ES} \times \{ \textit{true}, \textit{false} \}$$

Informally, a query evaluation amounts to checking whether the goal is satisfied directly by inspecting the state of the system, and if this is not the case whether the goal can be satisfied by applying one or more of the rules of R, or by asking the user to supply the needed information.

Rule application can be pictured as $(\Sigma,R,\delta) \Rightarrow^{r}([\![r]\!](\Sigma),R,\delta)$ where $[\![r]\!]$ is the operator defined in the previous section. Whenever some goal g of the form *same d* is presented a collection of rules might be selected, say $R' \subseteq R$, and applying the rules in some order, say

$$(\Sigma,R,\delta) \Rightarrow^{r_1} \cdots \Rightarrow^{r_k} (\Sigma',R,\delta) \quad \text{with } r_1,\ldots,r_k \in R',$$

might suffice to derive the goal, that is to transform the state of the system in such a way that testing the condition corresponding to the goal delivers *true*.
However, instead of selecting the set R', and determining the order in which the rules have to be applied in one step, under a backward-chaining strategy the rules are selected dynamically. This is accomplished by selecting the rules that are useful with respect to a certain goal, determining the sets of subgoals for this goal and (recursively) trying to establish the subgoals. Cf. [VBM75]
Given a goal, whenever an action operates upon an item with an object-attribute pair similar to the goal pair, the rule containing the action is selected for application. Testing the conditions of a rule, then, modifies the state Σ, since each condition is treated as a (sub) goal, for which possibly rules have to be applied.

Conforming to these remarks the definitions that follow describe the procedure of rule evaluation as mapping an expert system into an expert system, resulting in a sequence of expert systems growing with respect to the information contained in the Σ-state and decreasing in the number of pairs for which further rule applications are allowed.

The selection of rules is done by looking at the actions of rules and to check whether an action corresponds to some goal. However, not an identity-check is performed, as might be sufficient in a pure propositional system, but the comparison involves only the *⟨object,attribute⟩* pairs of the goal and the actions.

DEFINITION 4. 4: *rule-selection*

To enable the selection of applicable rules the function ρ_c^{π}, dependent on a condition c and a projection function π, is defined by

$$\rho_c^{\pi}(R) = \{ r \in R \mid r \equiv c_1,\ldots,c_n \rightarrow ac_1,\ldots,ac_m, \quad \text{and for some } i, 1 \leqslant i \leqslant m, \ \pi(ac_i) = \pi(c) \}$$

□

An extra complication is caused by the possibility to dynamically add facts. In order to model the *user-system dialog* we define the function *ask* by

DEFINITION 4. 5: $ask(\langle o,a\rangle) = \{\, add\ \langle o,a,v\rangle \mid v$ a value supplied by the user $\}$

This function is used to collect the answers from a user with regard to an $\langle o,a\rangle$ pair. Note that the answer of the user is transformed to a set of actions as might occur in the conclusion of a rule.

The definition below shows how query and rule-evaluation transform the state of the system.
The interpreter can be in three modes γ-mode, to evaluate queries or test conditions, α-mode, to execute actions, and ρ-mode to apply rules. Unwanted recursion is prevented by recording the status of object-attribute pairs in the control function δ. Apart from the status for each object-attribute pair, also the global status of the inference will be kept in the function δ. This is achieved by extending the domain of δ with *control* that, for this simple case, can only take the value *true* or *false*.

Before describing the interpreter we need the auxiliary

DEFINITION 4. 6:

(*i*) For arbitrary function σ the function variant $\sigma\{\, v/x \,\}$ is defined by

$$\sigma\{\, v/x \,\}(y) = \text{ if } x = y \text{ then } v \text{ else } \sigma(y) \text{ fi}$$

(*ii*) Function composition $g \circ f$ is to be understood as $\lambda x.g(f(x))$. □

DEFINITION 4. 7: *interpreter*

1. The evaluation of conditions and queries:

Five cases can occur. When *control* has the value *false* then the evaluation has no effect. If *control* says *true* and the goal is satisfied because of the current state of the system then also no action is taken. If either of these is not the case then the *control* is set to *false* unless the goal is *askable* or *derivable*. In this case $[\![\ .\ ]\!]_\alpha$ or $[\![\ .\ ]\!]_\rho$ are applied, after modifying the status of the object-attribute pair of the goal. Note that the goal is tested again, by means of γ, after the dialog or rule application has taken place; and *control* is modified by the result of this test.

$$[\![\ c\ ]\!]_\gamma(\Sigma,R,\delta) = \begin{cases} (\Sigma,R,\delta) \text{ if } \delta(control) = false, \text{ else} \\ (\Sigma,R,\delta) \text{ if } \gamma(c)(\Sigma)\text{, else} \\ (\Sigma',R,\delta'\{\, \gamma(c)(\Sigma')/control \,\}) \quad \text{if } \delta(\pi(c)) = askable\text{, else} \\ (\Sigma'',R,\delta''\{\, \gamma(c)(\Sigma'')/control \,\}) \quad \text{if } \delta(\pi(c)) = derivable\text{, and} \\ (\Sigma,R,\delta\{\, false/control \,\}) \text{ otherwise} \end{cases}$$

where

$$(\Sigma',R,\delta') = [\![\ ask(\pi(c))\ ]\!]_\alpha(\Sigma,R,\delta\{\, derived/\pi(c) \,\})$$
$$(\Sigma'',R,\delta'') = [\![\ \rho_c^\pi(R)\ ]\!]_\rho(\Sigma,R,\delta\{\, derived/\pi(c) \,\})$$

2. The evaluation of actions:

There are only two possibilities. If the control is set to *false* then the action has no effect. Otherwise the actions are just executed.

$$[\![\ ac\ ]\!]_\alpha(\Sigma,R,\delta) = \begin{cases} (\Sigma,R,\delta) \text{ if } \delta(control) = false \\ (\alpha(ac)(\Sigma),R,\delta) \text{ otherwise} \end{cases}$$

3. The evaluation of rules:

For convenience a rule $c_1,\ldots,c_n \rightarrow ac_1,\ldots,ac_m$ is given as $C \rightarrow A$ with $c_i \in C$, $i = 1,\ldots,n$, and $ac_j \in A$, $j = 1,\ldots,m$. The sets C and A are considered ordered such that a selection-function σ can choose the least element. The value of *control* is set to *true* before the application of a rule, since the failure of previous rules might have caused *control* to be *false*.

$$[\![\ C \rightarrow A\ ]\!]_\rho(\Sigma,R,\delta) = [\![\ A\ ]\!]_\alpha \circ [\![\ C\ ]\!]_\gamma(\Sigma,R,\delta\{\, true/control \,\})$$

4. For arbitrary sets of conditions, actions and rules it is assumed that there exist selection

functions σ_i, with $i \in \{\gamma,\alpha,\rho\}$, for selecting elements from these sets, and functions $rest_{\sigma_i}$ defined by $rest_{\sigma_i} = S \setminus \{\sigma_i(S)\}$. The empty set works as an identity.

$$[\![\ \varnothing\ ]\!]_i(\Sigma,R,\delta) = (\Sigma,R,\delta)$$
$$[\![\ S\ ]\!]_i(\Sigma,R,\delta) = [\![\ rest_{\sigma_i}(S)\ ]\!]_i([\![\ \sigma_i(S)\ ]\!]_i(\Sigma,R,\delta))$$

REMARKS:

- Instead of putting the *control* in δ it also could have been put in a seperate parameter. It is obvious that for a non-sequential execution of the system the control-part would be considerably more complex.
- The selection-functions are assumed to be given beforehand. Normally they correspond with the order of the conditions actions and rules as they occur in the knowledge base. The selection-function σ_γ for the conditions determines which subgoal will be expanded. The selection-function σ_ρ determines which rule will be applied. The definition above assumes exhaustive rule application. Non-exhaustive rule application can be modeled by redefining $rest_{\sigma_\rho}$ in an obvious way.

EXAMPLES:

In these examples p,p',p'' will be used for propositions. The indications *same* and *add* are omitted since they are clear from the context where p,p' and p'' occur.

(*i*) Let $KBS = (\{p\},R)$ with $R = \{p \rightarrow p'\}$, $\Sigma = \chi(\{p\})$ and δ such that *control* = *true* and p' *derivable*. Note that $\Sigma(p) = true$ and $\Sigma(p') = false$.

$$[\![\ p'\ ]\!]_\gamma(\Sigma,R,\delta) = (\Sigma',R,\delta'\{\,true/control\,\})$$

which follows from

$$\begin{aligned}[\![\ \{p\rightarrow p'\}\ ]\!]_\rho(\Sigma,R,\delta') &= [\![\ p\rightarrow p'\ ]\!]_\rho(\Sigma,R,\delta') \\ &= [\![\ p'\ ]\!]_\alpha([\![\ p\ ]\!]_\gamma(\Sigma,R,\delta')) \\ &= [\![\ p'\ ]\!]_\alpha(\Sigma,R,\delta') \\ &= (\Sigma',R,\delta')\end{aligned}$$

with

$$\Sigma' = \alpha(p')(\Sigma)$$
$$\delta' = \delta\{\,derived/\pi(p')\,\}$$

Hence, $\Sigma'(p) = \Sigma'(p') = true$.

(*ii*) Let $KBS = (\{\},R)$ with $R = \{r_1{:}p\rightarrow p', r_2{:}p\rightarrow p'', r_3{:}p'\rightarrow p''\}$, $\Sigma = \chi(\varnothing)$, and δ such that *control* = *true* and p, p' and p'' *derivable*.

$$[\![\ p''\ ]\!]_\gamma(\Sigma,R,\delta) = (\Sigma,R,\delta''\{\,false/control\,\})$$

which follows from

$$\begin{aligned}[\![\ r_3\ ]\!]_\rho([\![\ r_2\ ]\!]_\rho(\Sigma,R,\delta)) &= [\![\ r_3\ ]\!]_\rho([\![\ p''\ ]\!]_\alpha \circ [\![\ p\ ]\!]_\gamma(\Sigma,R,\delta'\{\,true/control\,\})) \\ &= [\![\ r_3\ ]\!]_\rho([\![\ p''\ ]\!]_\alpha(\Sigma,R,\delta'\{\,false/control\,\})) \\ &= [\![\ p''\ ]\!]_\alpha \circ [\![\ p'\ ]\!]_\gamma(\Sigma,R,\delta'\{\,true/control\,\}) \\ &= \cdots \\ &= (\Sigma,R,\delta''\{\,false/control\,\})\end{aligned}$$

with

$$\delta' = \delta\{\,derived/\pi(p'')\,\}$$
$$\delta'' = \delta'\{\,derived/\pi(p')\,\}$$

Since Σ hasn't changed, $\Sigma(p'') = false$.

REMARK: The definition above states how δ acts as a filter to avoid unwanted recursion. It will be shown how this can be achieved directly by filtering the rules.

Say R^F is the result of *filtering R through F* for $F \subseteq O \times A$ if R^F is obtained from R by removing every rule of the form $c_1,...,c_n \rightarrow ac$ if $\pi(ac) \in F$ and replacing every rule of the form $c_1,...,c_n \rightarrow ac_1,...,ac_m$ by a rule in which all ac_i, $1 \leq i \leq m$, are removed for which $\pi(ac_i) \in F$, and the entire rule is removed if no action remains. Now simply replace in the definition of condition evaluation $[\![\rho_c^{\pi}(R)]\!]_{\rho}(\Sigma,R,\delta\{ derived/\pi(c) \})$ by $[\![\rho_c^{\pi}(R)]\!]_{\rho}(\Sigma,R^{\{\pi(c)\}},\delta)$.

Query evaluation is defined as a mapping QE that takes an expert system and a query into a (possibly) modified system and a truth-value.

DEFINITION 4. 8: The mapping $QE : \underline{ES} \times G \rightarrow \underline{ES} \times \{ true, false \}$ is defined by

$$QE(\Sigma,R,\delta)(c) = ((\Sigma',R',\delta'),\delta'(control))$$

where $(\Sigma',R',\delta') = [\![c]\!]_{\gamma}(\Sigma,R,\delta)$. □

A simple property of QE is stated in

LEMMA 4. 9: Let $(\Sigma',R',\delta') = [\![\rho_c^{\pi}(R)]\!]_{\rho}(\Sigma,R,\delta\{ derived/\pi(c) \})$, then

$$QE(\Sigma,R,\delta)(c) = ((\Sigma',R',\delta'), true) \text{ iff } \gamma(c)(\Sigma') = true$$

PROOF: Immediate from the definition. □

To state some further properties of the functions constituting the interpreter we need

DEFINITION 4. 10

1. Let $|\delta| = \{p \mid \delta(p) = derivable, p \in O \times A\}$, call $|\delta|$ the cardinality of δ, and say that the cardinality of δ is less than the cardinality of δ', $\delta \leq \delta'$, whenever $|d| \subseteq |\delta'|$.
2. Associate with each expert system the set $|(\Sigma,R,\delta)| =_{df} |\Sigma|$ of currently known facts. □

Now we can state

LEMMA 4. 11: Assume $Pred = \{ same \}$ and $Act = \{ add \}$, then for an arbitrary rule r

1. $[\![r]\!]_{\rho}$ is monotonic in the sense that for Σ, Σ' with $|\Sigma| \subseteq |\Sigma'|$

$$|[\![r]\!]_{\rho}(\Sigma,R,\delta)| \subseteq |[\![r]\!]_{\rho}(\Sigma',R,\delta)|$$

2. $[\![r]\!]_{\rho}$ is conservative in the sense that for any Σ

$$|\Sigma| \subseteq |[\![r]\!]_{\rho}(\Sigma,R,\delta)|$$

3. For $(\Sigma',R',\delta') = [\![r]\!]_{\rho}(\Sigma,R,\delta)$ it holds that $\delta' \leq \delta$.

PROOF: Use the fact that γ is monotonic and that α is both monotonic and conservative. Also observe that $\delta' < \delta$ whenever a condition in the antecedent of r is not established by merely looking in the set of facts or asking the user. □

The consultation of an expert system results in a sequence of expert systems increasing in information content, as contained in the state, and decreasing in the opportunities for deriving facts. To provide some intuition for the definitions given above, the intermediate states resulting from $[\![r]\!]_{\rho}(\Sigma,R,\delta)$, the evaluation of a rule r in a state Σ with control information δ, will be characterized as such a sequence.

The notation $(\cdots)_i$ is used to project to the i-th component of a tuple.

Let $ES_0,ES_1,...,ES_k$ be the sequence defined by

$$ES_0 = (\Sigma_0, R, \delta_0) = (\Sigma, R, \delta)$$
$$ES_{i+1} = (\Sigma_{i+1}, R, \delta_{i+1})$$

for $i = 0,..,n-1$

$$\Sigma_{i+1} = ((QE(\Sigma_i, R, \delta_i)(c_{i+1}))_1)_1$$
$$\delta_{i+1} = ((QE(\Sigma_i, R, \delta_i)(c_{i+1}))_1)_3$$

where c_i is the i-th condition in $r \equiv c_1,...,c_n \rightarrow ac_1,...,ac_m$, and $k = n+m$, and for $i = n,...,k$

$$\Sigma_{i+1} = \alpha(ac_{i-n+1})(\Sigma_i)$$
$$\delta_{i+1} = \delta_i$$

Let $tt_0 = true$ and $tt_{i+1} = tt_i \wedge (QE(\Sigma_i, R, \delta_i)(c_{i+1}))_2$ for $i = 0,...,n-1$.
Let j be the least i between 0 and $n-1$ such that $tt_i = false$ and let $j = n$ if no such i exists.
Then

$$[\![r]\!]_\rho(\Sigma, R, \delta) = \text{ if } j = n \text{ then } ES_{n+m} \text{ else } ES_j \text{ fi}$$

Moreover, assuming that when $[\![r]\!]_\rho(\Sigma, R, \delta) = ES_j$ with $j<k$ that $ES_{i+1} = ES_i$ for $i+1>j$, it holds that $|\Sigma_i| \subseteq |\Sigma_{i+i}|$ for $i = 1,...,k-1$, and $\delta_{i+1} \leqslant \delta_i$. Alternatively, applying a filtering of the rules by defining $R_{i+1} = ((QE(\Sigma_i, R_i, \delta_i)(c_{i+1}))_1)_2$ then $R_{i+1} \subseteq R_i$. Note that each element ES_i of the sequence $ES_1,...,ES_j$ might be the result of a sequence $ES_{i_1},...,ES_{i_j}$ due to the fact that the evaluation of c_i necessitates the application of a rule.

4.2. Completeness

Looping can safely be prevented by simply keeping a set of goals on which no further expansion is allowed. See [LR81]. In the expert system interpreter described, however, a less selective criterium both for the selection of rules and the loop-trap is used, namely not the occurrence of an item of the form $\langle o,a,v \rangle$ but the occurrence of the $\langle o,a \rangle$ part. This has an immediate drawback in terms of the restrictions that are to be imposed on the rule-base with respect to the dependencies between *⟨object,attribute⟩* pairs.

DEFINITION 4. 12: *⟨o,a⟩-acyclicity*

Say that $\langle o,a \rangle$ is dependent on $\langle o',a' \rangle$ if for some rule $r \in R$ the consequent of r contains an action $ac \equiv act\ \langle o,a,v \rangle$ and the antecedent of r contains a condition $c \equiv pred\ \langle o',a',v' \rangle$. The rule-base R is called *⟨o,a⟩-acyclic* if for no $o \in O$ and $a \in A$ the dependency relation is cyclic. □

Intuitively, the activity of the interpreter corresponds to building a proof tree, by means of a goal reductive process. At the leaves of the tree are the items given as initial facts or items that resulted from asking the user. The intermediate nodes are labeled with atomic propositions that are true on behalf of the subsequent nodes. The root of the tree represents the goal to be proven.
When trying to construct a proof tree for a goal by expanding a node one can check whether the node expanded is equal to a node on the path to the root. This procedure does not affect soundness or completeness when the nodes are atomic propositions. Cf. [ABW86,LR81] The proof trees obtained in this way are *regular* in the sense that no proposition occurs more than once on a path from the root to one of the leaves. The loop-trap performed by the function δ however is less selective in that it forbids any $\langle o,a \rangle$-pair to occur more than once on a path from the root to one of the leaves. This restricts the set of allowed proof trees for a particular item to the proof trees that are *strictly regular*, in the sense that no $\langle o,a \rangle$ pair occurs more than once on a path. This gives rise to an incompleteness, unless the restriction of $\langle o,a \rangle$-acyclicity is imposed on the rule-base. An example of this will be given after the theorem claiming the soundness and completeness of the procedure.

The following theorem relates the set of facts computed by a bottom-up procedure as defined in section 2 and 3, under a certain restriction with respect to the structure of the proof trees for these facts, to the results delivered by the top-down interpreter defined in this section. The proof of the theorem needs an auxiliary lemma that establishes the independence of the value of *control* in δ of the order of evaluating conditions.

LEMMA 4. 13:
Let $(\Sigma',R,\delta') = [\![c_1]\!]_\gamma \circ [\![c_2]\!]_\gamma(\Sigma,R,\delta)$ and $(\Sigma'',R,\delta'') = [\![c_2]\!]_\gamma \circ [\![c_1]\!]_\gamma(\Sigma,R,\delta)$
then it holds that $\delta'(control) = true$ *iff* $\delta''(control) = true$.

PROOF: Assume the contrary, and let for instance $\delta''(control) = false$. Then the evaluation of c_1 blocks the derivation of items that are needed to establish c_2. This can only occur if for some p, say, evaluating c_1 sets p to *derived*.

· Suppose $\delta(p) = askable$, then all the values for p must have been asked during the evaluation of c_1. Since the user is not allowed to change his mind, and actions cannot be undone, evaluating c_2 could not have resulted in different values for p. Contradiction.

· Now suppose $\delta(p) = derivable$, then the assumption would force us to believe that applying rules for p in the context of evaluating c_1 would give less information than applying them in the context of evaluating c_2. But then the rules for p would have to contain conditions with the same concern as c_1 or subgoals of c_1, a situation which is precluded by the requirement of $\langle o,a \rangle$-acyclicity. □

THEOREM 4. 14: *completeness*

Let $S_\delta = \cup\{ d \mid (add\ d) \in ask(p), \delta(p) = askable \}$ then, for R $\langle o,a \rangle$-acyclic, there is a finite m such that $QE(\Sigma,R,\delta)(same\ d)$ delivers *true iff* $d \in |[\![R]\!]^m(\Sigma \cup \chi(S_\delta))|$, provided that whenever not $\Sigma(d) = true$ then $\delta(\pi(d))$ is *derivable* or *askable*.

PROOF: The theorem amounts to saying that an item will be affirmed *iff* a finite number of forward rule applications suffices to derive the item such that the proof tree for that item is strictly regular.
For abbreviation let $\chi_\delta = \chi(S_\delta)$ and $\Sigma_\delta = \Sigma \cup \chi_\delta$.
($\Rightarrow$) The proof is by induction on the cardinality of δ.

· If $|\delta| = \varnothing$ then it must be the case that d is either given or asked and hence $d \in |\Sigma_\delta|$.

· When $|\delta| \neq \varnothing$, take as induction hypothesis: for $\delta' < \delta$ there is a m such that $d \in |[\![R]\!]^m(\Sigma_{\delta'})|$ whenever $QE(\Sigma,R,\delta')(same\ d)$ delivers *true*. Only the case that d is derived by the application of the rules in $\rho_c^\pi(R)$ will be considered. If d was derived in this way then there must be some rule $r \equiv c_1,...,c_n \rightarrow ...,ac,...$ with $ac \equiv add\ d$ for which it holds that $d \in |[\![r]\!]_\rho(\Sigma,R,\delta')|$ with $\delta' = \delta\{ derived/\pi(d) \}$. Now the proof hinges on the possibility to arbitrarily change the order of evaluation of the conditions in the rule whenever that rule succeeds. Applying the lemma above gives us that $QE(\Sigma,R,\delta')(c_i)$ delivers *true*, $i = 1,...,n$, and, by induction, that $\gamma(c_i)([\![R]\!]^m(\Sigma_{\delta'}))$. By the definition of $[\![\cdot]\!]$ it then holds that $d \in |[\![r]\!]_\rho([\![R]\!]^m(\Sigma_{\delta'}))| \subseteq |[\![R]\!]^{m+1}(\Sigma_{\delta'})|$. Since δ' differs from δ only in that $\pi(d)$ is set to *derived* it holds that $|\chi_{\delta'}| = |\chi_\delta|$.
($\Leftarrow$) With induction on m it has to be proved that if $d \in |[\![R]\!]^m(\Sigma_{\delta'})|$ then
$QE(\Sigma,R,\delta)(same\ d)$ delivers *true*.

· For $m = 0$ the result is immediate, since then d is given or asked.

· Let $m > 0$ and assume that d is derived by rule $r \equiv c_1,...,c_n \rightarrow ...,ac,...$, then $d \in |[\![r]\!]([\![R]\!]^{m-1}(\Sigma_\delta))|$ and $\gamma(c_i)([\![R]\!]^{m-1}(\Sigma_\delta)) = true$, $i = 1,...,n$. By induction it follows that $QE(\Sigma,R,\delta)(c_i)$ delivers *true*, $i = 1,...,n$, which gives us that $\gamma(c_i)([\![c_i]\!]_\gamma(\Sigma,R,\delta))$. Due to the condition that $\delta(\pi(d)) = derivable$ r can be applied. By the fact that $[\![\cdot]\!]_\rho$ is conservative and γ monotonic, d will be affirmed in the state resulting from the application of r and possibly other rules. Hence $QE(\Sigma,R,\delta)(same\ d)$ delivers *true*. □

REMARKS:

· Suppose that R is not $\langle o,a \rangle$-acyclic and let $It = \{ p \}$ and (omitting predicate- and action-

symbols) $R = \{p \rightarrow p', p' \rightarrow p''\}$ with p, p', p'' containing as items only $\langle o,a \rangle$-variants. Then the proof tree for p'' would be $p'' \leftarrow p' \leftarrow p$, however, since by the application of $p' \rightarrow p''$ $\delta(\pi(p')) = \delta(\langle o,a \rangle) = \delta(\pi(p''))$ becomes *derived*, the rule $p \rightarrow p'$ is not applicable, making $\gamma(p')(\chi(It))$ false. Some subtleties, however, from the fact that a rule may contain more than one action. Let $R' = \{(s \rightarrow p), (p \rightarrow s,p'), (s,p' \rightarrow p'')\}$, then the evaluation for the query p'' would succeed due to the side-effect of applying the rule $p \rightarrow s,p'$ selected for the condition s. Taken as Horn sentences R' is equivalent to $\{(s \rightarrow p), (p \rightarrow s), (p \rightarrow p'), (s,p' \rightarrow p'')\}$. Note that $|[\![It,R]\!]| \subseteq |[\![It,R']\!]|$, indicating that the trick to circumvent the restriction is to enrich the knowledge-base with useless information.

· Note that the maximal depth of a proof tree cannot exceed the total number of *⟨object,attribute⟩* pairs. Without the restriction of $\langle o,a \rangle$-acyclicity the upperbound on the depth is determined by the size of the domain of discourse D. Moreover, with regard to efficiency, it can be remarked that, since all derived subgoals are stored, no work is redone; and since the procedure is top-down only a minimum of superflous information is generated.

When restricting the predicate-symbols to $\{same\}$ and the action-symbols to $\{add\}$ the only difference between an expert system, applying rules to facts, and a logic system, evaluating ground Horn sentences, is that an expert system keeps a record of items generated during the inference. A loop-trap checking for repeated subgoals has been modeled by a (less selective) function δ containing the status-information (necessitating the requirement of $\langle o,a \rangle$-acyclicity). A simplifying assumption inherent in the definition of δ is that the user is supposed to supply all relevant values for a given pair either when asked for or as initial data.

The implementation of a simple expert system corresponding to the characterization given in this section is described in [P0112].

5. Extensions

The system described in the previous section is still a toy-system in the sense that it covers only a small part of the functionality of an expert-system (shell) like EMYCIN [BS84]. These systems allow many more predicates and actions, comparisons between values, and the derivation of negative information. In this section the framework developed previously will be adapted to cope with some of these extensions. To be able to still characterize the meaning of a knowledge base in terms of (pseudo) deductive systems it has to be required that the operator corresponding to the rules behaves conservative in the sense that it only adds information, positive or negative. It should be remarked that the production systems commonly used to implement expert systems do not conform to this requirement, because of the fact that on each cycle of rule-application information gathered previously might be destroyed. Simply gathering information, however, has the drawback that an inconsistent state might result. The solution to this problem is to regard the inference as a two stage process, one for gathering information and one for checking the reliability of the result by means of some arbiter.

From the treatment in the previous sections it will be apparent that the predicate *same* and the action *add* can safely be eliminated without affecting the meaning of the rules. The action *add* can simply be seen as a directive to the implementation to store a value for the object-attribute pair.
Actually *predicate* seems to be a misnomer. It is more natural to view the attribute as a predicate concerning an object and a value. The role of the predicate *same*, however, becomes clear only in a context where certainty-factors are used. Then one has the choice between asking for *same* $\langle o,a,v \rangle$ when one is interested only in a certainty-factor greater than say 0.2 and *definite* $\langle o,a,v \rangle$ when one requires that the certainty-factor is exactly 1, meaning that object o has value v for attribute a with absolute certainty.
Also the system provides for a predicate *known*, as in *known* $\langle o,a \rangle$, to make the satisfaction of a condition of a rule dependent on the fact whether a certainty-factor has been established for some value

of $\langle o,a \rangle$.
This use suggests that these predicates are to be viewed as a kind of meta predicates, allowing to enquire after the status of the inference.
The use of certainty-factors moreover introduces the possibility to derive negative information.
Another refinement is that for each attribute one can indicate whether it is *single-valued* or *multi-valued*, meaning, resp., that the attribute can acquire only one value with the exclusion of others, or several, mutually non-exclusive values.
These observations suggest to extend the knowledge representation language in the following way.

DEFINITION 5. 2: *knowledge representation*

1. For each $\langle o,a \rangle$ pair it can be indicated whether it is single-valued or multi-valued
2. The set *Pred* is extended to include the predicates *same*, *notsame* and *known*.
   The set *Act* is extended to include the actions *affirm* and *deny*.

REMARKS:
· The requirement of singlevaluedness imposes the constraint that only a single value will be derived for that pair. If more than one value is derived the result will be considered unreliable.
· The action *deny* is included to allow the derivation of negative information. This action, unlike an action as *delete*, does not destroy information. Some interesting remarks with respect to the treatment of negation can be found in [Ga88]. Instead of the action-symbol *add* the action-symbol *affirm* is used.
· The meta-predicate *known* is of a completely different nature than the other (meta) predicates. It is used primarily to test whether any value has been derived for an attribute. If this is not the case then the test fails and the rule-application is further abandoned.

In order to reason both with positive and negative information the domain of facts has to be enriched as to include whether a statement has been affirmed or denied, and similarly for the domain of values.

DEFINITION 5. 3: *augmented domains*

1. For $S \subseteq D$ let $S^+ =_{df} \{ \langle o,a,v,true \rangle \mid \langle o,a,v \rangle \in S \}$
   and $S^- =_{df} \{ \langle o,a,v,false \rangle \mid \langle o,a,v \rangle \in S \}$
2. Let $\overline{V} =_{df} V \times \{ true, false \}$ and $\overline{V}^*$ the set of finite subsets of $\overline{V}$.
3. The projection ν is defined by $\nu(\langle o,a,v \rangle) =_{df} v$.

REMARK: Part (1) introduces augmented facts, in order to cope with negative information. The augmented values introduced in (2) correspond with a four-valued logic with truthvalues *unknown*, *true*, *false* and *contradictory* associated with respectively, for each value v, with $\varnothing$, $\{(v,true)\}$, $\{(v,false)\}$ and $\{(v,true),(v,false)\}$, ordered by the ordering induced by set-inclusion. Such a logic has been introduced by Belnap [Be77] to cope with the presence of unreliable information, and has also been applied by Sandewall [Sa85] to deal with non-monotonic inferences.

DEFINITION 5. 4: *state*

Let $\Sigma^* = \{ \Sigma \mid \Sigma : Pair \rightarrow \overline{V}^* \}$ be the set of valuations delivering for each pair a set of augmented values.
For each augmented state Σ the set associated with Σ is

$$|\Sigma| =_{df} \{ \langle o,a,v,t \rangle \mid (v,t) \in \Sigma(\langle o,a \rangle) \}$$

For states Σ and Σ' the inclusion $\Sigma \subseteq \Sigma'$ denotes $|\Sigma| \subseteq |\Sigma'|$, and $\Sigma \cup \Sigma'$ denotes $\chi(|\Sigma| \cup |\Sigma'|)$ where

$$\chi(S)(\langle o,a\rangle) = \{(v,t) \mid \langle o,a,v,t\rangle \in S\}$$

□

Note that a state of knowledge is defined in such a way that it accomodates the wish to reason with positive and negative information, and at the same time allows a check whether the constraint of single-valuedness is respected, by testing the cardinality of the set of augmented values that belong to the attribute in question.

In our opinion there is a simple syntactic criterium for inconsistency, namely the occurrence of both an affirmation and a denial of the statement that an attribute has some value.

DEFINITION 5. 5: *reliability*

To check the reliability of the information in a state Σ we need:

$$\Sigma\text{-}consistent(\langle o,a\rangle) =_{df} \neg\exists v \in V.(v,true) \in \Sigma(\langle o,a\rangle) \wedge (v,false) \in \Sigma(\langle o,a\rangle)$$

□

To check whether a state Σ is reliable we have to test for each pair p if Σ-*consistent*(p) and when p is singlevalued if the cardinality of the set of true items in $\Sigma(p)$ is not greater than one.

For the definition of the interpretation that is to be attached to the (meta) predicate- and action-symbols there is the choice between a monotonic extension, that is an extension that does not affect the monotonic behaviour of the operator associated with the rules and one that does make the associated operator non-monotonic. First the monotonic extension will be given and then some comments will be given to characterize the non-monotonic extension.

DEFINITION 5. 6: *evaluating conditions and actions*

The meaning of (meta) predicate-symbols and action-symbols can be defined by

$$\gamma(same\ \ d)(\Sigma) = (\nu(d),true) \in \Sigma(\pi(d))$$
$$\gamma(notsame\ \ d)(\Sigma) = (\nu(d),false) \in \Sigma(\pi(d))$$
$$\gamma(known\ \ d)(\Sigma) = \Sigma(\pi(d)) \neq \varnothing$$
$$\alpha(affirm\ \ d)(\Sigma) = \Sigma\{\ \{(\nu(d),true)\} \cup \Sigma(\pi(d))/\pi(d)\}$$
$$\alpha(deny\ \ d)(\Sigma) = \Sigma\{\ \{(\nu(d),false)\} \cup \Sigma(\pi(d))/\pi(d)\}$$

REMARK: An implementation can decide when to check whether the information added by the actions compromises the reliability of the system.

For this extension it holds that γ is monotonic and that α is both monotonic and conservative. If however a meta predicate *notknown* would have been added or the definition of *notsame* would have been changed as in

$$\gamma(notsame\ \ d)(\Sigma) = (\nu(d),false) \in \Sigma(\pi(d)) \vee (\nu(d),true) \notin \Sigma(\pi(d)$$
$$\gamma(notknown\ d)(\Sigma) = \Sigma(\pi(d)) = \varnothing$$

then γ would no longer be monotonic. As a remark, note that positive and negative information are not symmetrically dealt with. When introducing negation one can for instance have that *not same* $\langle o,a,v\rangle \neq$ *notsame* $\langle o,a,v\rangle$. However this is only the case if $\Sigma(\langle o,a\rangle)$ contains both $(v,true)$ and $(v,false)$, but then, because of inconsistency, anything is allowed.
The monotonic extension can still be characterized as a deductive system, that is as an operator transforming a set of positive and negative statements taken from $D^+ \cup D^-$ to another (unique) set of these statements. The non-monotonic extension, however, can only be characterized as a pseudo-deductive system in that having more information could preclude that conclusions are drawn whereas

in the absence of this information these conclusions are simply drawn.

6. Conclusions

The theory of deductive systems provides a context for describing the capabilities of a rule-based expert system. In order to apply the theory of deductive systems it had to be guaranteed that the actions occurring in the rules preserve the information contained in the state in which these rules are allowed to fire. A detailed description has been given of the strategy of inference employed by the family of diagnostic expert systems. The results produced by the inference engine have been characterized as a certain kind of proof trees, and it has been shown that the procedure is not complete, with respect to the intended meaning of the knowledge base, unless the rules conform to certain restrictions.

References

[ABW86] K. Apt , H. Blair and A. Walker *Towards a theory of declarative knowledge* In: Foundations of Deductive Databases and Logic Programming J. Minker (ed.) Morgan Kaufmann, Los Altos, 1987

[Be77] N.D. Belnap *A useful four-valued logic* In: Modern uses of multiple-valued logic J.M. Dunn and G. Epstein (eds.) Reidel Dordrecht, 1977, pp. 5-37

[Be87] M. Bezem *Consistency of rule-based expert systems* CWI Report CS-R8736

[BK88] R.M. Butler and N.T. Karonis *Exploitation of parallelism in prototypical deduction problems* CADE-9 LNCS 310 Springer, 1988, pp. 333-343

[BS84] B.G. Buchanan, E.H. Shortliffe (eds.) *Rule Based Expert Systems* Addison Wesley 1984

[Bu83] *Expert Systems* Butler Cox Foundation Sept. 1983

[Cl74] J.P. Cleave *The notion of logical consequence in the logic of inexact predicates* Zeitschr. f. math. Logik und Grundl. d. Math. Bd. 20, 1974, pp. 307-324

[Co85] P.R. Cohen *Heuristic reasoning about uncertainty: an Artificial Intelligence approach* Pitman Boston, 1985

[Ga88] D.M. Gabbay *What is negation in a system?* Logic Colloquium 1986 F.R. Drake and J.K. Truss (eds.) Elsevier North-Holland, 1988, pp. 95-112

[JM84] N.Jones and A. Mycroft *Stepwise development of operational and denotational semantics for Prolog* 1984 Int. Symp. on Logic Programming, Atlantic City, 1984, pp. 281-288

[LM83] J.-L. Lassez and M.J. Maher *The denotational semantics of Horn clauses as production-systems* in: Proc. AAAI-83 Washington, 1983, pp. 229-231

[LS76] D.W. Loveland and M.E. Stickel *A hole in goal-trees: Some guidance from resolution-theory* IEEE C-25 April 1976, pp. 335-341

[LR81] D.W. Loveland and C.R. Reddy *Deleting Repeated Subgoals in the Problem-reduction format* JACM Vol. 28 No. 4, October 1981, pp. 646-661

[Lu86] P.J.F. Lucas *Knowledge representation and inference in rule based expert systems* CWI Report CS-R8613

[Ma87] S. Maslov *Theory of Deductive Systems and its applications* Foundations of Computing MIT Press 1987

[PS85] P.F. Patel-Schneider *A decidable first-order logic for knowledge representation* In: Proc. IJCAI 1985, pp. 455-458

[P0112] A. Eliëns *A simple expert system in POOL* Prisma document P0112, 1987

[P0298] A. Eliëns *A simple parallel expert system* Prisma document P0298, 1988

[Sa85] E. Sandewall *A functional approach to non-monotonic logic* Comput. Intell. 1, 1985, pp. 80-87

[St77] M. Stickel *Fuzzy four-valued logic for inconsistency and uncertainty* Conference on Multivalued Logic Chicago 1977, pp. 91-94

[Ta56] A. Tarski *Logic, Semantics, Metamathematics* Clarendon, Oxford 1956

[VBM75] G.J. VanderBrug and J. Minker *State-Space, Problem-reduction and Theorem Proving - Some relationships* CACM 1975 Vol. 18 No 2, pp. 107-115