THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

How to Improve the Reliability of Expert Systems

Citation for published version:
Bundy, A 1987, How to Improve the Reliability of Expert Systems. in S Moralee (ed.), *Proceedings of Expert Systems '87 on Research and Development in Expert Systems IV*. Cambridge University Press. <http://books.google.co.uk/books?hl=en&lr=&id=lXu9zXv3FiQC&oi=fnd&pg=PA65&dq=how+to+improve+the+reliability+of+expert+systems&ots=uAuCBOUKjD&sig=VwUNutOtPGHw4jUVLGzTBg5bgQQ#v=onepage&q=how%20to%20improve%20the%20reliability%20of%20expert%20systems&f=false>

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Proceedings of Expert Systems '87 on Research and Development in Expert Systems IV

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 14. Jul. 2024

How to Improve the Reliability of Expert Systems

Alan Bundy

Abstract

Reliability is likely to become an increasingly important issue for knowledge engineers. Without assurances of reliability they will be restricted in the scale and scope of the expert systems they can build and will not be able to capitalise on their current success. The key to greater reliability is a sound theoretical foundation for current and new knowledge engineering techniques — making knowledge engineering a proper engineering science. We illustrate the kind of theoretical work that is required with a few case studies, in the areas of search control, fault diagnosis and learning.

Acknowledgements

I am grateful for conversations on the topics of this paper with Richard O'Keefe, Aaron Sloman, Robert Inder, Roberto Desimone, Bill Sharpe and Mike Uschold. I would also like to thank Bill Sharpe and Hewlett Packard for allowing me to work on this paper in the ivory tower of their Bristol Research Laboratory away from the hubbub of university life. Some of the work described here was funded by SERC grant GR/44874.

Keywords

Expert systems, knowledge engineering, reliability, logic, AI techniques, basic AI, engineering science.

1 Introduction

The UK expert system community has been very successful in the development of small scale, commercial, rule-based, expert systems. A typical example is a fault diagnosis system for a piece of specialised hardware, consisting of a set of less than 100 rules, running on a PC, in one of the many commercial shells. Part of the success consists of the unexpected (to me anyway) discovery of a large number of commercially interesting problems which yield to such a simple mechanism.

UK knowledge engineers have also been active in building much larger expert systems, with hundreds or even thousands of rules. In addition, they have experimented with alternative knowledge representation and reasoning techniques, e.g. frames, objects, semantic nets, etc. This use of large scale expert systems and of alternative and/or multiple knowledge representations has been more typical of the US market, but both are becoming more important here.

I pay tribute to the success of UK knowledge engineering, but in this paper I want to look beyond it and see what we need to do now in order to lay a solid foundation for future success. To see what might be required in the future we must first explore the limitations of existing systems. The main limitation I will focus on is the, unreliability of current expert systems. This is an important topic in its own right, but assumes greater importance in the context of the shift to both larger scale and to alternative and mixed knowledge representations. Unreliability often increases as the system gets larger; large collections of rules are more likely to be inconsistent, incomplete or to interact in unpredicted ways, because it is harder for humans to check them 'by hand'. Unreliability is also likely to increase when alternative knowledge representation formalisms are used, because these new formalisms are often poorly understood or are combined in poorly understood ways.

2 Unreliability

What do we mean by the term *unreliable* as applied to an expert system? It is a catch-all term, and can include any of the following overlapping phenomena.

- **Fragility** (non-robustness): The system may fail in unexpected ways.
- **Unpredictability**: The user either cannot specify the circumstances under which the system will produce an answer or cannot specify the type of answer that will be produced.
- **Brittleness** (non-flexibility): The system cannot deal with problems on which it has not been previously tested.
- **Discontinuity**: The system gives very different output in response to similar input.

All of these phenomena are undesirable in a commercial product. In a real time situation, unreliability can cause chaos. In a life critical situation, unreliability can be fatal. Even in more mundane situations, unreliability can cause expensive mistakes and lead to users rejecting the product. The unreliability of expert system shells and toolkits makes it difficult for a knowledge engineer to decide whether a particular tool will help solve the current problem; thus it is difficult to advise customers whether their problems are soluble by current techniques or to predict whether a knowledge engineering project will be successful. Thus reliability is a very desirable feature both for suppliers and users of expert systems.

Contrast the situation of expert systems with more mature branches of engineering. Before building a bridge, a structural engineer makes a number of drawings and carries out elaborate calculations. These calculations enable the engineer to predict the behaviour of the bridge under a wide variety of stresses. As a result the engineer can say, with confidence, that for a specific range of loading and weather conditions the bridge will behave robustly, predictably, flexibly and continuously. If the engineer does not carry out these calculations or does so inaccurately, and the bridge falls down, then he/she is guilty of professional negligence (*cf* the Tay Bridge disaster).

3 Causes of Unreliability in Expert Systems

There is not, yet, the same tradition in the field of knowledge engineering. In fact, there is a common assumption that analogous assurances of reliability could not be given; that algorithms do not exist on which analogous calculations could be based; that the heuristics on which expert systems are based lead to an inherently unreliable product.

I will argue that this is wrong. It is possible to base expert systems upon techniques which lend themselves to the giving of reliability assurances. Some work of this kind already exists and I will outline it below.

However, it is true that existing knowledge engineering practice leads to unreliable products. For instance, the following practices are of that kind.

- The mixing of control and factual information in the same rule.
- The attachment of arbitrary procedures to rules.
- The use of multiple knowledge representation formalisms without a clear understanding of their relationship to each other.
- The use of "uncertainty factors" without a clear understanding of their meaning.
- The incremental development of systems by patches and hacks, and without consideration of the whole system.

- A lack of theoretical understanding of the techniques used in the system's development.

The last of the these points, *a lack of theoretical understanding*, is a generalization of the others. We will use it below as a summary of the causes of unreliability.

This kind of bad practice is encouraged by existing expert system shells, toolkits and knowledge representation systems. For instance, most expert system shells provide a facility for attaching numbers to rules and facts to represent "uncertainty", but few explain what these numbers mean. Many alternative interpretations are possible. This may cause the numbers to be used differently by different knowledge engineers and users, leading to a clash of expectations about the expert system and, hence, to unreliability.

Suppose an expert system for crime detection is investigating the murder of Mary. When it asks a witness whether John hated Mary, the witness can choose a range of options between 1, meaning "no", and 5, meaning "yes". What would an answer of 3 mean? The system's designer may have intended 3 to mean a mild form of hate. The rules of the system will embody this meaning and draw appropriate inferences from it. However, the witness might use 3 to mean that someone hated Mary intensely, but that it might not have been John. Consequently, the expert system may fail to suggest a culprit for Mary's murder or may suggest an innocent person.

Most toolkit systems (e.g. ART, KEE, LOOPS, etc) offer a range of different knowledge representation formalisms, e.g. rules, frames, objects, semantic nets, isa hierarchies, procedures, but no account of the relationship between them. The user of the toolkit is faced with the difficult choice of choosing an appropriate representational formalism. The representational scopes of the formalisms overlap considerably. For instance, a relationship of membership between an element and a set may be represented as a predicate in a logical assertion, an arc in an isa hierarchy or a slot in a frame. However, the toolkit may not be capable of translating between equivalent representations. So if a mixture of formalisms is used the toolkit may not be able to combine the knowledge effectively. More confusing still, the toolkit may translate between the formalisms only some of the time, so that the user is unable to predict when two pieces of knowledge will be effectively combined and when they will not, [Inder 87b]. At the root of the problem is a lack of theoretical understanding of the relationship between the formalisms, for instance, is the slot in a frame merely syntactic sugar for a logical assertion or is it something subtly different?

4 Resolution: A Model of Propriety

Among this collection of, mostly unreliable, knowledge-engineering techniques, one family stand out as a model of respectability and reliability: the techniques of logical deduction used in automatic theorem proving and logic programming, e.g. resolution. The reliability of resolution-like, deduction techniques is given by their semantics, and by the soundness and completeness theorems that accompany these semantics. How does this work?

The semantics of a logical formalism is a systematic way of assigning meaning to its expressions. The semantics of predicate calculus (*aka* first order logic) was invented by Tarski. Since then semantics have been given for other types of logic, e.g. by Kripke for modal logics. Each symbol of the logic is of a particular kind, e.g. constants, variables, functions, predicates, etc. An *interpretation* of predicate logic consists of a set of objects, called the *universe*, and some *mappings* on this universe. The constants of the logic refer to particular objects in the universe, the variables range over the objects, the functions refer to mappings from objects to objects, and the predicates refer to mappings from objects to truth values. Interpretation independent mappings are associated with the logical connectives and quantifiers, e.g. the implication arrow, $\longrightarrow$; the conjunction symbol, &; and the universal quantifer, $\forall$. Tarskian semantics uses these mappings to calculate the references of complex expressions from the references of their sub-expressions.

So, for instance, we might choose the set of real world objects as the universe. We might associate with the predicates *gun* and *weapon*, mappings from each object, o, in this universe to the set $\{true, false\}$ such that $gun(o)$ is *true* if and only if o is a gun and $weapon(o)$ is *true* if and only if o is a weapon. Under this interpretation the complex expression:

$$\forall X \; gun(X) \longrightarrow weapon(X)$$

is forced to have the reference *true*, since in the real world all objects that are guns are also weapons.

An expert system rule or fact can be regarded as a formula of (predicate) logic if it can be interpreted as having a (Tarskian) semantics. In a knowledge base, consisting of a set of such logical formula, we would obviously want them all to have the same semantics. It is up to the author of a knowledge base to say what universe its variables range over, what its constants refer to, and what mappings to associate with its functions and predicates. A reference can then be calculated for each formula in it. The author of the knowledge base will almost certainly want the reference of each formula in it to be *true*. An interpretation in which each member of a knowledge base is *true*, is called a *model* of that knowledge base. If a formula is *true* in every model of a knowledge base, then it is called a *logical consequence* of that knowledge base.

The pay-off of this investment is that any formula, C, deduced by resolution from a knowledge base, K, will be a logical consequence of K. This is the soundness theorem of resolution. In addition, every logical consequence of K can be deduced by resolution from it. This is the completeness theorem of resolution.

What this means to the user of a resolution-based expert system shell is that it is possible to have some weak form of prediction about the behaviour of the system. To get the benefit the user must try to write *true* rules and facts according to some specific assumptions about what objects exist in the universe and what the other symbols in the knowledge base mean. The user should add sufficient rules and facts to the knowledge base so that only the intended interpretations are models of all of them. The shell will then be guaranteed to draw all and only those conclusions which are true in all these model. This does not prevent the user being surprised at some of the conclusions which are true in these models, but it does curtail some of the wilder flights of inference to which expert systems are prone.

The user can also use resolution to test if the knowledge base is inconsistent. If it is inconsistent then it will have no model, i.e. there will be no circumstance in which all the rules and facts are simultaneously *true*. In this case resolution is guaranteed to be able to deduce *false* from them. So to test for inconsistency, resolution can be set loose on the knowledge base: if it deduces *false* then the knowledge base is inconsistent; if it does not, then the knowledge base is consistent. Unfortunately, there is no time limit to how long this might take - you could wait a long time and still not be sure whether *false* has just not been deduced yet or never will be.

The theory underlying resolution thus makes possible some procedures, analogous to the bridge builder's calculations, which enable certain predictions to be made about the behaviour of an expert system. Consider what reliability assurances are made possible by these calculations.

- **Robustness:** The system will not fail to deduce a conclusion if it is a logical consequence of the knowledge base.
- **Predictability:** The user can specify that the system will deduce a conclusion if and only if it is a logical consequence of the knowledge base.
- **Flexibility:** The system can deduce any logical consequence of the knowledge base, even though the user has not tested this deduction in advance.
- **Continuity:** If two similar knowledge bases both share the same models, then the same conclusions will be deduced from them.

Thus the burden of reliability assurance is shifted from the procedure of deduction to the semantic concept of logical consequence. These assurances fall a long way short of what we would like. They still leave a large burden on the user in designing a knowledge base with just the intended models and, hence, just the intended logical consequences. They give no assurances about the amount of effort required to deduce a logical consequence. In addition, they only apply to a situation in which logical deduction is being applied to a knowledge base. They do not apply to other kinds of reasoning, e.g. learning or inference involving uncertainty However, they are a start. In the next section we consider how they might be strengthened and extended to other kinds of reasoning.

5 Does this Idea Generalise?

In order to develop similar kinds of reliability assurance for expert systems based on other knowledge engineering techniques we must first develop similar kinds of theoretical frameworks. But before we can do this we have to define the other knowledge engineering techniques in question. The resolution family of logical deduction techniques is not just unusual in having a clear theoretical framework, but is also unusual in being clearly defined.

Basic AI research can be seen as the attempt to develop a loose collection of computational techniques of which resolution is one, [Bundy 86]. New techniques are often developed using exploratory programming, and AI researchers have not been diligent in separating them clearly from the program in which they were invented, nor in analysing their theoretical properties, nor in using such an analysis to extend and generalise them. The Catalogue of Artificial Intelligence Techniques, [Bundy 84], is an attempt to identify, informally define, and catalogue AI techniques. Mathematical logic is one of the most promising mathematical tools for the formal definition, analysis and extension of these informally defined techniques.

A major aspect of the formal analysis of knowledge engineering techniques is the classification of the different kinds of knowledge used in them. For instance, when using exploratory programming, knowledge engineers often include both factual and control knowledge in the same rule. However, a resolution-style analysis of the semantics of the rule is concerned only with its factual content; in order not to confuse the analysis it is necessary to separate off the control knowledge, e.g. into a separate meta-level (see section 5.1 below). This separation buys considerable computational advantages; it makes possible a simpler semantics which simplifies the tasks of the automatic inference and learning of both factual and control knowledge, [Bundy 81].

Similarly, a theoretical analysis of uncertainty cannot be given without separating the different kinds of uncertainty that the knowledge engineer may entwine into the same value. In the example above we saw that an uncertainty value of 3 applied to the fact "John hated Mary" is ambiguous; it could be modifying either its intensity or the identity of the hater (or hatee). If the fact were represented by the logical formula $hate(John, mary)$[1] then the ambiguity can be represented by considering the uncertainty value as applying either to the predicate *hate* or to the constant *john* (or *mary*).

In addition to all this, there are the more obvious ambiguities about just what the value 3 means in terms of reduction of intensity or doubts about someone's identity, etc. Sometimes this kind of ambiguity can be clarified by interpreting an uncertainty as a probability. For instance, a value of 3 (out of 5) applied to the rule:

$$hate(M, V) \ \& \ possess(M, W) \ \& \ weapon(W) \longrightarrow kill(M, V)$$

might mean that person A will kill person B in 60% of the cases in which A hates B and possesses

[1] In this paper we adopt the Prolog convention that words starting with a capital letter represent variables and those that do not represent constants, so we have to represent the constant "John" as "*john*".

a weapon C. Note that this probabilistic interpretation is not available if the uncertainty value is taken as a measure of the intensity of the hate or the doubt about the identity of someone.

Teasing apart the different kinds of knowledge used in expert systems makes it easier to develop procedures for reasoning with the different kinds of knowledge. We are then in a position to strengthen existing reasoning procedures and to develop new ones. We are also in a position to classify users's tasks according to the kind of knowledge they involve and give a prediction about which techniques, if any, are appropriate to their solution.

The sort of work that needs to be done is best illustrated by examples of existing work employing this methodology. In the following sections I will give some examples of such work. This is not meant to be an exhaustive list: it is limited by available space and my own ignorance.

5.1 Proof Plans for Guiding Inference

A major problem with automatic inference techniques is the *combinatorial explosion*: the number of possible inference paths that an expert system must explore grows exponentially or worse with the number of inference steps. When attempting a non-trivial inference, the expert system becomes bogged down in the possibilities. The usual answer to the combinatorial explosion is the provision of a heuristic, search control mechanism. A heuristic is a rule of thumb which is not guaranteed to work, but often does. Heuristics suggest which paths to explore first. For instance, if you are trying to prove a conjecture in which there are two or more occurrences of a term then at some point you must apply a rule or fact which reduces those occurrences to one or zero. Such rules can be identified syntactically and it is worth trying to apply them before other rules.

The design and selection of heuristics has been relegated by most workers to the realm of 'art form'. This is unfortunate. The provision of good heuristics is crucial to the success of an expert system afflicted with the combinatorial explosion, i.e. any system whose rules create many choice points during inference and in which the inference is deep. Without such heuristics the expert system will be unable to draw the required conclusion within a reasonable time, but will be bogged down in unpromising parts of the inference process. In order that the user can predict whether the expert system is going to succeed in a reasonable time, it is necessary to promote the provision of search control from 'art-form' to 'science'.

My own research group has been exploring how this might be done. We have chosen the domain of mathematics because it contains many deep and highly branching, deductive inferences, which cause large combinatorial explosions. However, experienced mathematicians have developed sophisticated techniques for controlling their search for a proof.

We have tried to represent this search control knowledge using a logical formalism, which we call a *meta-logic*, with its own declarative semantics. To use it to control the search for a proof we make deductions with it; a process that we call *meta-level inference*, [Bundy 81]. The universe of the meta-logic consists of the expressions and proof strategies of the area of mathematics in which we are trying to prove theorems: the *object theory*. Meta-level inference analyses the object-level conjecture to be proved and finds an appropriate proof strategy, which it then applies.

Recently, we have been trying to capture the concept of a proof plan using these techniques. Mathematicians, when trying to prove a conjecture, often have a plan of how to proceed. Such a proof plan should have the following properties:

- **Usefulness**: The plan should control the search for a proof.
- **Expectancy**: The use of the plan should carry some expectation of success.
- **Incertitude**: On the other hand, success cannot be guaranteed. (Most interesting mathematical theories have no decision procedure. A proof plan which was always successful would amount to a decision procedure.)

- **Name:** $collection(Term)$
- **Input:** $Before$
- **Output:** $After$
- **Preconditions:** $occ(Term, Before) = B$
- **Effects:** $occ(Term, After) = A \;\&\; A < B$
- **Tactic:** a program for applying collection rules and facts

Figure 1: Proof Method for Collection

- **Patchability:** If the plan should fail it should be possible to patch it by providing alternative steps for the failing ones.

If our representation of proof plans can capture the correct balance between 'expectancy' and 'incertitude', then we can provide a tool for predicting whether an expert system will succeed in a reasonable time. Necessarily, such a tool would not be perfect, but it would give as good a prediction as it was possible to give, and it could be used to localise the possible causes of failure.

We see a proof strategy as consisting of a program of proof tactics, [Gordon 79], where each proof tactic is a procedure for performing a small part of the proof, e.g. collecting several occurrences of an equation unknown into one, [Bundy 81]; unfolding a recursively defined function, [Darlington 81]; applying mathematical induction. Note that some tactics may fail, causing failure of the whole strategy. For instance, a tactic for collecting two occurrences of a term into one may work by applying a rule drawn from a suitable set. Thus the two occurrences of the term *john* in the goal $hate(john, john)$ may be collected by applying the rule:

$$depressed(D) \longrightarrow hate(D, D)$$

backwards to produce $depressed(john)$. On the other hand, the attempt to collect the *johns* in $possess(john, john)$ might fail because no suitable rule was available.

A proof method is a meta-level specification of a proof tactic. It contains a *precondition* and an *effect*. Both are descriptions in the meta-logic: the precondition describes the expression that the tactic applies to, and the effect describes the expression that the tactic produces if it succeeds. An example of the method for collection is given in figure 1.

A proof plan is a meta-level specification of a proof strategy, i.e. it is a kind of super-method. It is so constructed that the preconditions of each of its sub-methods are either implied by its preconditions or by the effects of earlier sub-methods. Similarly, its effects are implied by the effects of its sub-methods. The original conjecture should satisfy the preconditions of the plan; the effects of the plan should imply that the conjecture has been proved. Executing a proof plan consists of running each of its tactics according to the program it specifies. A proof plan can either be hand coded by the system builder, or the techniques of automatic program synthesis can be used to construct it.

This representation meets the specification given above, point by point, as follows:

- **Usefulness:** As the tactics run they will each perform a part of the object-level proof.
- **Expectancy:** If the conjecture meets the preconditions of the plan and each tactic succeeds then the effects of the plan will be true and the conjecture will be proved.
- **Incertitude:** However, a tactic may fail, causing failure of the plan.

- **Patchability:** Since the preconditions and effects of a failing tactic are known, program synthesis techniques may be (re)used to patch the gap in the plan with a subplan.

5.2 Abduction for Diagnosing Causes of Effects

In the diagnosis of faults in machines or diseases in people, we are interested in working from the observed symptoms to discover their underlying causes. This process may be applied recursively: working from symptoms to their immediate causes, then to the causes of these causes, and so on until we reach something we can regard as a fault/disease.

Since Mycin, the classic way to automate this process has been by using production rules of the form:

IF $effect_1$ is present
AND $effect_2$ is present
$\vdots$
AND $effect_n$ is present
THEN *cause* is present with likelihood c

In Mycin, and many other expert systems, these rules are used backwards, i.e. from suspected diseases to observed symptoms. This is a little technical detail that is irrelevant to the discussion below, but has the potential to confuse it totally. Please pay attention only to the direction of the chain of implications, which is from effects to causes, even if this chain is constructed in reverse.

IF/THEN production rules like this seem very similar to logical implication rules. Translating the above production rule into the logical rule:

$$effect_1 \,\&\, effect_2 \,\&\, \ldots \,\&\, effect_n \longrightarrow cause \tag{1}$$

seems like a good way to put it onto a firm theoretical foundation.

Unfortunately, the semantics of this logical rule gives us some trouble. The causal link runs in the opposite direction; causes cause effects, not vice versa. This suggests that the logical implication also runs from right to left, that is, if a particular cause is present then some particular combination of effects will also be present. This would be represented, logically, as:

$$cause \longrightarrow effect_1 \,\&\, effect_2 \,\&\, \ldots \,\&\, effect_n \tag{2}$$

It is not at all obvious that rule 1 holds, i.e. that the presence of this combination of effects implies the presence of this cause. The effects may be due to some other cause or combination of causes.

For instance, if we think that a cause of person B possessing an object P is that B bought P then the type 1 representation is:

$$possess(B, P) \longrightarrow buy(B, P)$$

whereas the type 2 representation is:

$$buy(B, P) \longrightarrow possess(B, P)$$

There is clearly something wrong with the type 1 representation; it is possible to possess something without buying it, for instance someone might give it to you or you might steal it.

This lack of logical implication in rules of type 1 is one reason why, from Mycin onwards, knowledge engineers have felt it necessary to attach uncertainty values to production rules; the uncertainty expresses the lack of strict logical implication from the effects to the causes. If we use

logical implications running in the opposite direction, from causes to effects, then we can dispense with such uncertainty values [2].

Unfortunately, if we use implications running from causes to effects then we cannot use deduction (as Mycin effectively does) to deduce what faults/diseases are causing the symptoms observed. The process of conjecturing hypotheses that might imply a given fact is called *abduction*, [Kowalski 79]. But for a formula to be regarded as the fault/disease causing a symptom, it is not enough that the fault/disease logically imply the symptom; some additional conditions must be met. Cox and Pietrzykowski, [Cox 86], have recently specified some candidate additional conditions, which define what they call *fundamental* causes. Here are their definitions:

C is a cause of E with K, where K is some knowledge base, if and only if:

- $C \;\&\; K \longrightarrow E$ must hold, i.e. C together with K must 'account for' E.
- C is a formula, with no free variables, consisting of some quantifiers followed by a conjunction of negated and unnegated propositions. The idea is that C represents some concrete situation. We do not want to count as a cause anything that is a vague, so we do not want to allow disjunction, for instance. Compare the concept of mental model from cognitive science, [Inder 87a].

C is a fundamental cause of E with K if and only if it is a cause of E with K and it is:

- **Consistent:** $C \;\&\; K$ must have a model. We do not want C to be a cause of E just because it is inconsistent with K and, therefore, the conjunction of them implies everything [3].
- **Non-trivial:** $C \longrightarrow E$ does not hold. We do not want the connection between C and E to be a tautology, we want it to involve some knowledge from K in an essential way.
- **Basic:** Every consistent cause of C is trivial. We do not want C to itself have a cause. This condition makes C be the ultimate cause, e.g. the actual fault or disease.
- **Minimal:** For all causes, C' of E, $C \longrightarrow C'$ implies $C \longleftarrow C'$. We do not want C to have any extraneous material; we want it to be the simplest formula consistent with the other conditions.

In addition, Cox and Pietrzykowski have developed a procedure for conjecturing fundamental causes given their effects. This works with rules of the form 2, and is an alternative fault/disease diagnosis technique to the Mycin one of using deduction on rules of the form 1. The basic idea of the Cox-Pietrzykowski abduction procedure is to apply resolution to $K \;\&\; E$ and look for dead-ends in the search space. These dead-ends are sure to be basic causes. They are then processed in various ways to ensure that they are also consistent, non-trivial and minimal. This procedure may fail to terminate, either in the generation of dead-ends or in the test for consistency. However, if it does terminate then the soundness and completeness of resolution can be used to show that it generates all and only the fundamental causes of an effect. It can also be easily adapted to find basic causes that fail to be fundamental, e.g. are not consistent or minimal.

For example, let K be the knowledge base:

(a) $hate(M,V) \;\&\; possess(M,W) \;\&\; weapon(W) \longrightarrow kill(M,V)$
(b) $depressed(D) \longrightarrow hate(D,D)$
(c) $buy(B,P) \longrightarrow possess(B,P)$
(d) $gun(G) \longrightarrow weapon(G)$

[2] Unless, of course, we require them for some other reason, e.g. to express partial ignorance about the effect of a cause or the cause of an effect.

[3] In more recent work this condition has been relaxed somewhat, but the new condition is rather more complex, and so I have omitted it from this elementary discussion.

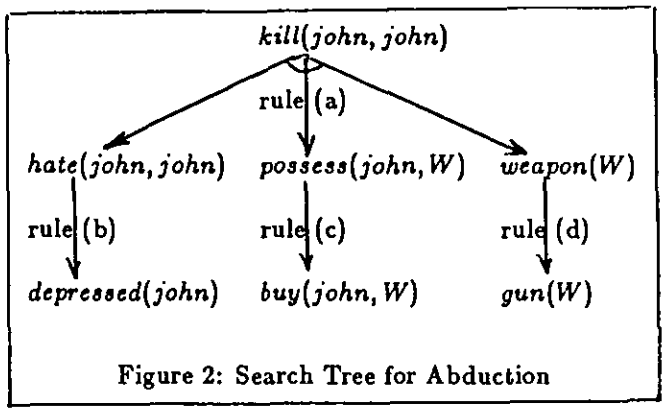

Figure 2: Search Tree for Abduction

Let E be the symptom $kill(john, john)$, i.e. John commits suicide. To find the fundamental cause of this suicide, we first run a process of deduction on E using K. This generates the search tree given in figure 2.

The last line of figure 2 is, taken together, a dead-end. It is consistent with K, non-trivial and minimal. It needs to be processed into the logical form of causes to give:

$$\exists W \; depressed(john) \;\&\; buy(john, W) \;\&\; gun(W)$$

i.e. the fundamental cause of John's suicide is that he was depressed and bought a gun. This would not satisfy a psychiatrist, but it is the poverty of the knowledge base that is at fault not the abduction procedure

This theoretically based work on abduction puts it on a similar footing to deduction. The user of the abduction procedure has some account of what the procedure finds, i.e. fundamental causes, and some assurance that it will find them. When it fails to do so it fails for good reasons, i.e. the essentially undecidable nature of the problem, and not because of some kludge in the program.

We have seen that one form of uncertainty has been taken out of the diagnosis process: the uncertainty associated with the lack of logical implication in the Mycin-type rules. As a result the fundamental causes produced by the abduction procedure are not ordered by uncertainty values. If meaningful uncertainty values could be associated with the rules then the abduction procedure could easily be adapted to incorporate this 'feature'. These uncertainty values would be essentially 'cleaner' than the Mycin ones in that they would not be associated with the lack of logical implication, but would reflect only the remaining sources of uncertainty, e.g. ignorance about the precise causes of effects, the probability of an effect following a cause, or whatever.

The only ordering of causes implicit in the abduction procedure is a partial one generated by the conditions met by each cause, e.g. a fundamental cause is better than one that is not minimal. It would be interesting to experiment with additional conditions: including some domain specific ones, and consider the orders implied by them. For instance, some logical expressions might be preferred as causes either because they describe faults more likely to occur or because they describe more concrete situations.

It would also be interesting to classify the rules in the knowledge base according to whether they represented causal links, and to invent additional conditions in the definition of fundamental cause requiring causal and non-causal rules to be used differently, e.g. in the deduction of E from C and K. For instance, rules (a) and (c) above seem to express a causal link, whereas (b) might be considered part of the definition of *depressed* and (d) is taxonomic information. The derivation of E ought to use at least one causal rule. Possibly, there are other requirements.

5.3 Explanation-Based Learning

One of the most exciting areas of recent research in automatic learning is the development of explanation-based learning (EBL) techniques. EBL techniques learn a target concept by generalising an example. This involves separating the incidental features of the example from the essential ones. Similarity-based learning techniques do this by comparing many examples and discovering the features common to all of them. In contrast, EBL techniques usually work from a single example of a piece of reasoning in which the target concept plays a central rôle.

A number of recent researchers have put explanation-based learning on a formal basis, [Mitchell 86,KedarCabelli 87,DeJong 86]; in particular, they have given an account of EBL in terms of logical deduction. We summarise this below.

Mitchell has stated the EBL problem as follows:

Given:

- **Target Concept:** A high level description of the concept to be learned, but at the wrong level of description.
- **Training Example:** A description of a particular example of the concept at the right level of description.
- **Domain Theory:** What we have above called a knowledge base. This can be used to relate the right to the wrong levels of description via a deductive chain.
- **Operationality Criteria:** A description of what counts as the right level of description, typically the listing of 'ok' predicates, etc.

Determine:

- **Partial Definition:** A partial definition of the target concept at the right level of description.

The basic idea of the procedure is to find a deductive chain linking the training example to its partial definition at the right level of description, to generalise this chain abstracting away the example specific aspects, and then to use this generalised chain to construct the required definition of the target concept.

An example will illustrate this. We will use the same knowledge base as for the example in section 5.2 above[4]. Suppose the training example is:

(e) $depressed(john)$
(f) $buy(john, gun1)$
(g) $gun(gun1)$

and the target concept is $kill(X, Y)$, i.e. we want to learn part of the operational definition of killing from a particular example of killing.

The first step is use the knowledge base to discover why the training example is an example of killing. This is done by deducing

$$\exists X, Y \; kill(X, Y)$$

from the knowledge base and the training example. The derivation is represented by the proof tree in figure 3. The arcs of the proof tree are labelled with both the rule or fact names and the substitutions required to apply the rules. Read T/X as T is substituted for X.

[4]In fact, this knowledge base was first developed by DeJong, [DeJong 86], to illustrate EBL.

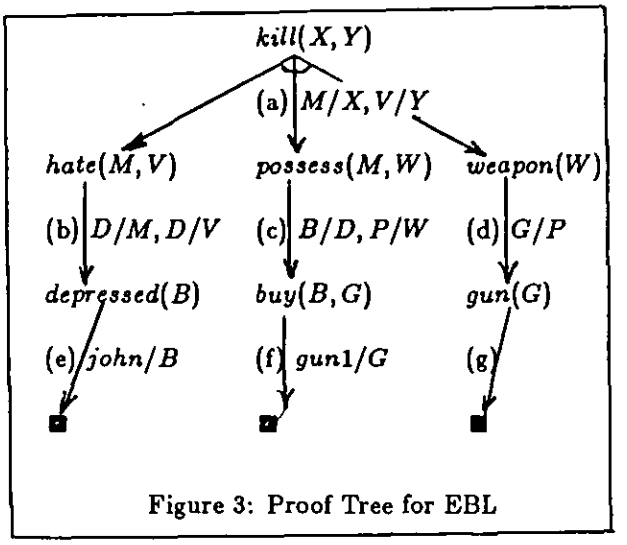

Figure 3: Proof Tree for EBL

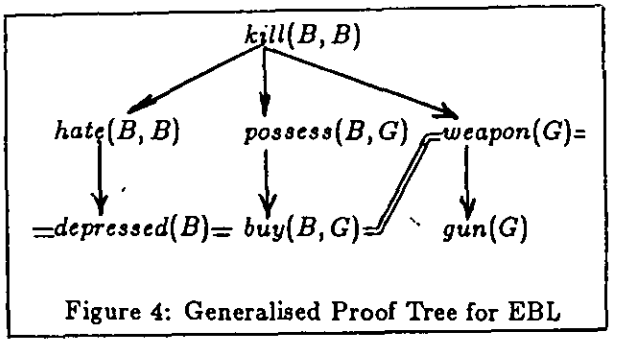

Figure 4: Generalised Proof Tree for EBL

The second step is to generalise this proof tree. Any uses of the training example and substitutions associated with such use are deleted from the tree. The remaining substitutions are applied to the formulae in the tree. This gives the generalised tree in figure 4.

The third step is to apply the operationality criteria to decide which predicates in the tree are at the right level. Suppose we decide to allow: *depressed*, *buy*, *gun* and *weapon*. We must choose a horizontal slice through the tree which involves only these predicates. Such a slice is marked by a double line in figure 4. We then conjoin the formulae joined by this line and form an implication between them and the formula at the root of the tree. This gives:

$$depressed(B) \;\&\; buy(B,G) \;\&\; weapon(G) \longrightarrow kill(B,B)$$

which is the partial definition of the target concept required. In fact, it is a partial definition of a special case of the killing concept, namely suicide.

This theoretically based work on EBL gives both a clear definition of the purpose of this learning technique and some assurances about the reliability of the procedure. The partial definition of the target concept is an implication between a formula meeting the operationality criteria and an instance of the target concept. This implication is a logical consequence of the knowledge base; it is formed by unifying a chain of rules from the knowledge base and in programming terms is an example of *partial evaluation*, [Safra 86]. The soundness theorem of resolution gives the required guarantee that the partial definition is a logical consequence of the knowledge base. The completeness theorem gives a guarantee that any logical consequence of the appropriate logical form can be produced by the EBL procedure, in particular, that an appropriate training example

can be defined to produce each such logical consequence.

6 Conclusion

In this paper I have argued that knowledge engineering needs to live up to its name and become a proper engineering science: providing the kinds of assurances of reliability that are normal in other branches of engineering. The way to do this is to put the techniques used to build expert systems onto a sound theoretical basis. The tools of mathematical logic appear to be a good basis for doing this, but we need to be imaginative in their use — not restricting ourselves to deductive inference, but investigating other aspects of reasoning: search control, abduction, learning, uncertainty, alternative knowledge representation techniques, etc.

This theoretical work is normally seen as being the province of academic researchers, but it need not be exclusively so. In any case, the success of the theoretical work is of crucial importance to applied researchers, in industry and elsewhere; without it they will not be able to build reliable expert systems.

The interaction is not all in one direction; applied researchers have important feedback to provide on the utility of existing knowledge engineering techniques 'in the field'. Basic researchers are particularly interested in the *failures* of current techniques, so that they may be improved. Unfortunately, such failures are usually poorly described in the literature, if they are described at all. We need a change of attitude among knowledge engineers, so that the failure of an expert system is not seen to reflect badly on its designers, but to be a valuable result of an experiment with particular knowledge engineering techniques.

'Failure' is usually taken to mean the failure of a particular technique to be useful for a particular task. I also want to draw attention to failures of reliability, e.g. the user's failure to predict just what tasks the technique can succeed on. We will need to know about both kinds of failure if we are to build the reliable, large-scale, wider-ranging, expert systems of the future.

References

[Bundy 81] A. Bundy and B. Welham. Using meta-level inference for selective application of multiple rewrite rules in algebraic manipulation. *Artificial Intelligence*, 16(2):189–212, 1981. Also available as DAI Research Paper 121.

[Bundy 84] A. Bundy, editor. *Catalogue of Artificial Intelligence Tools*, Springer-Verlag, 1984. Second Edition.

[Bundy 86] A. Bundy. *What kind of field is AI?* Research Paper 305, Dept. of Artificial Intelligence, Edinburgh, 1986. To appear in the Proceedings of the Workshop on the Foundations of Artificial Intelligence, New Mexico, 1987.

[Cox 86] P.T. Cox and T. Pietrzykowski. Causes for events: their computation and applications. In J. Siekmann, editor, *Lecture Notes in Computer Science: Proceedings of the 8th International Conference on Automated Deduction*, pages 608–621, Springer-Verlag, 1986.

[Darlington 81] J. Darlington. An experimental program transformation and synthesis system. *Artificial Intelligence*, 16(3):1–46, August 1981.

[DeJong 86] G. DeJong and R. Mooney. Explanation-based learning: an alternate view. *Machine Learning*, 1(2):145–176, 1986.

[Gordon 79] M.J. Gordon, A.J. Milner, and C.P. Wadsworth. *Edinburgh LCF - A mechanised logic of computation.* Volume 78 of *Lecture Notes in Computer Science*, Springer Verlag, 1979.

[Inder 87a] R. Inder. *The Computer Modelling of Syllogistic Reasoning using Restricted Mental Models.* PhD thesis, Dept. of Artificial Intelligence, Edinburgh, 1987.

[Inder 87b] R. Inder. The state of the ART. *Airing*, (1):8–14, February 1987. Available from AI Applications Institute, Edinburgh.

[KedarCabelli 87] S. Kedar-Cabelli and L.T. McCarty. Explanation-based generalisation as resolution theorem proving. In P. Langley, editor, *Proceedings of the 4th International Machine Learning Workshop*, pages 383–389, Morgan Kaufmann, 1987.

[Kowalski 79] R. Kowalski. *Logic for Problem Solving. Artificial Intelligence Series*, North Holland, 1979.

[Mitchell 86] T.M. Mitchell, R.M. Keller, and S.T. Kedar-Cabelli. Explanation-based generalization: a unifying view. *Machine Learning*, 1(1):47–80, 1986. Also available as Tech. Report ML-TR-2, SUNJ Rutgers, 1985.

[Safra 86] S. Safra and E. Shapiro. *Meta Interpreters for Real.* Report CS86-11, Department of Computer Science, The Weizmann Institute of Science, May 1986.