COMPREHENSION BY MODEL-BUILDING AS A BASIS FOR AN EXPERT SYSTEM

J.L. Cunningham

Cognitive Science Research Paper

Serial no: CSRP 025

The University of Sussex
Cognitive Studies Programme
School of Social Sciences
Falmer
Brighton BN1 9QN

COMPREHENSION BY MODEL-BUILDING AS A BASIS FOR AN EXPERT SYSTEM

Abstract

This paper describes how a program designed to build a model (in the formal logical sense) can be used as the inference component of an Expert System. Some details of an existing program to do the model building task are described. This program works with a powerful many sorted logic. Use of a many sorted logic has computational advantages for this task, and these advantages also carry over to the use of the program as the inference component of an Expert System. In addition, some advanced features of the existing program form the basis for a non-numerical handling of uncertain knowledge.

1. Introduction

This paper describes the model building problem in section 1. It then briefly describes a sorted first order calculus of great expressive power, which is the formal language used by an existing model building program. This program is described in section 2.

Section 3 discusses the salient features of Expert Systems, and section 4 relates these features to the existing program described in section 2.

Further sections then go into greater detail on the major issues raised in section 4.

1.1 The Model Building Problem

This section starts with a very brief and informal description of the nature of a logical model. For a formal description of a model, see any introduction to symbolic logic, for example [Tapscott, 1976].

1.1.1

In what follows, the word "expression[11] is used loosely to mean either a term or formula, and "formula" a "well formed formula with no free (unquantified) variables". (The objection to free variables is not part of the theory, but is a personal preference.)

I will call a collection of logical formulae a theory.

If (and only if) formulae of a theory are mutually consistent, then it is possible to find a mapping of a certain kind from expressions of the logic into (the union of) two sets. One set is the set of the booleans, and the other set I will call the universe (this set can even be the set of expressions of the logical language). The mapping must be such that terms of the logic are mapped to elements of the universe, and formulae are mapped to the booleans. In particular, the formulae of the theory must be mapped to true. In addition, it should be possible to construct the value of the mapping for compound expresions from the values for atomic expressions in a way which corresponds to the meaning of the formulae. For example, if the mapping for two propositions P and Q is determined (either because they are atomic, or by applying this procedure recursively), then the disjunction of P and Q will be mapped to the boolean true if either or both of P and Q are mapped to Q, and to false otherwise. Similarly for other logical connectives, and for quantifiers.

Here is an example: suppose the universe includes three elements K, L, and ML If ^ and J^ are constant symbols in the logic, and j^ and ^ are mapped to the elements j≤ and L^ respectively of the universe, then mapping the terms f(k) and f(I) to the elements L^ and M^ respectively would require that f(f(k)) also be mapped to M^.

Such a mapping, together with its universe, is called a model (remember that the mapping must map formulae of the theory to true). If no model exists, then the formulae are inconsistent. A model for some collection of formulae is said to satisfy the formulae.

1.1.2

A model, in the sense used by this paper, can be viewed as a small database. This database must have two properties: firstly, all the facts in the database must satisfy a set of constraints, and secondly, the database must be complete enough that it would be impossible to add any new facts to the database which violated the constraints without directly contradicting facts already present in the database. To illustrate what I mean by "directly contradicting", and as an example, suppose that a database contains facts of the form "X is male" or "X is not male" or "X is female" or "X is not female" (where X stands for either of the names "Mary" or "Lesley"). Further, suppose that the only constraint is that someone can not be both male and female. Now, we will consider three possible databases:

(1) "Lesley is male"
"Lesley is female"

This could not be a model, because it violates the constraint.

(2) "Mary is female"
"Lesley is not female"

This does not violate any constraints but it is not a model because it is possible to add another fact, e.g. "Mary is male", which does not directly contradict a fact already in the database but does violate the constraint. However, it is possible to extend the database (2) to give:

(3) "Mary is female"

"Mary is not male"

"Lesley is not female"

"Lesley is male"

This database does not violate the constraint. Facts such as "Mary is male" which would violate the constraint are explicitly contradicted by facts already present in the database (in this case "Mary is not male".)

1.1.3

The model building problem is to find a model for a given consistent set of formulae (and to report failure if given inconsistent formulae). I shall modify this notion of a model in the model building program to be described in section 2. Under this modified formulation, the model building problem is to take a set of constraints and an initial database, and to extend the database until it is a model, as the example (2) was extended to give (3) (in section 1.1.2). Notice that it was necessary to restrict the possible values of "X" in the description of allowable facts, otherwise it would still have been possible to add facts to violate the constraints. For example we could have added the facts "John is male" and "John is female".

The effect of this modification is that I shall regard as a model a universe plus a representation of a mapping which is minimal in the sense that a full model could be generated from this model by the process described in section 1.1.1. For example,a minimal representation of the mapping for the example in section 1.1.1 would include explicitly a mapping from the symbols k and l, but would represent mappings from terms such as f(k) and f(f(k)) by mappings for expressions such as f(K) and f(L). (Note that these are not necessarily expressions of the

original logical language, since K and L need not be symbols of the language. I assume that if elements of the universe occur as constants in the language then they are always mapped onto themselves in any model.)

In a first order logic with only a finite number of non-logical symbols, if the universe of some model is finite, then the model itself is finite. For some sets of formulae no finite model exists - even though there may be infinite models. This possibility lends intuitive plausibility to the following (true) assertion "Determining whether an arbitrary set of logical formulae is consistent is not a computable problem (not decidable)".

> It is well known that only semi-decision procedures exist for first order logic, i.e. that it is possible to find a proof of a theorem if one exists, but that it is not a computable problem to determine whether a proof exists. Existing algorithms may not terminate for some formulae which are not provable. The assertion above follows from this, since if a theorem is not provable its negation is consistent.

Since the general problem is uncomputable, I have restricted myself to finding finite models for sets of formulae. For any given finite universe this problem can easily be shown to be equivalent to determining consistency in the propositional calculus, and is therefore decidable. However, for the problems of interest, the search space for most simple algorithms becomes enormous, and heuristic techniques must be used. In section 2, I will describe a program to do this task for formulae of a sorted logic.

The task of model-building must be contrasted with theorem proving,

although there is some overlap.

1.2 Sorted Logic

A sorted logic is one in which allowable models are restricted so that terms cannot be mapped to arbitrary elements of a universe. Instead, the universe is divided into subsets called sorts. Exactly how this division into sorts is used to restrict allowable models differs in different sorted logics. One common way to use sorts is to introduce quantifiers for each sort, so that whereas an unsorted formula:

(x) PCxD

might be read as "for all X, X is P", or more informally "everything is P", the corresponding sorted quantifier could be read as "for all X in sort S, X is P", or more informally as "every S is P". This last rendering immediately suggests that such a sorted logic might be reducible to an unsorted logic, by replacing the sorted formulae with corresponding unsorted formulae. The example could be replaced by:

(x) SCxl => PCx]

This suggested reduction of an axiom system in a sorted logic into an equivalent unsorted system, so that there is a one to one correspondence between the possible models for both systems, is always possible, by systematic transformations like that in the example above. There are also other systematic ways to achieve this reduction. However, a sorted logic has greater expressive power (is more concise), plus additional computational advantages (see section 5.1).

An alternative, which is equivalent to having sorted quantifiers, is to have sorted variables. The effect is much the same.

Most references to sorted logic assume that the sorts will be disjoint. Pat Hayes, in [Hayes, 1971], describes a particularly expressive sorted logic which allows non-disjoint sorts. In addition, instead of sorting the quantifiers or the variables explicitly, he proposes a sort function specifying the sort which the value of a functional expression may be in for each combination of sorts for its arguments. This sort function may be partial. By including predicates in this scheme, (and including booleans as part of the universe), he specifies that a formula is only well-formed if its sort can be determined. Variables are allowed to range over any values for which the formula is well-formed. It is a variant of this last form of sorted logic that is used by the program described in the next section. This logic is also used by A G Cohn [Cohn, 1983].

2. The Existing Program

A program has been built to build a model to satisfy a given (consistent) set of axioms. This section discusses some features of this program; these features are referred to in section 4 -- the discussion of using this program as the kernel of an expert system.

2.1 The Logic

2.1.1

The program uses a powerful sorted logic like that described at the end of section 1.2. This use of sorted logic allows a given problem to be axiomatised more concisely than in conventional unsorted logic, although it is necessary to specify the value of the sort function for the function and predicate symbols used in any particular problem. Even so, there is a net improvement in clarity and conciseness for all but the

most simple problems. (The program is designed to work with tens o axioms, rather than a few. For such problems the improvement in clari and conciseness is much greater than for only a few axioms.)

2.1.2

This representation language is used internally by the program, that i it is not reduced to some primitive normal form such as Horn clause All the usual logical connectives are allowed, plus some unusual on (for example, there is a "partitioning" connective, see section 2.1.4 Most connectives are not treated as binary connectives, but have natural n-ary form. Thus

(P <=> Q) and (Q <=> R) and (R <=> S)

may be written as

P <=> Q <=> R <=> S

If the propositions P, Q, R and S were themselves complex, there wou be an even greater computational saving in treating the equivalen connective as n-ary. (Consider also the representation of the abo expression in Disjunctive/Conjunctive normal form - even assuming t propositions are atomic.)

2.1.3

Function symbols as well as predicate and relation symbols may be use This allows a natural representation of one distinction between "th and "a" in English. Thus a Moslem might say:

Allah = god()

where a Hindu might say

God(Allah)

More specifically, the use of a function symbol is a convenient way to represent that there is only one of something.

This example also illustrates another unusual distinction made in the logic used by this program: there is a distinction made between constants and functions of no arguments. Informally, this distinction corresponds to the distinction between two kinds of singular definite noun phrases as made by, for example, Donnellan [Donnellan, 1966].

The constant symbol corresponds to a referential noun phrase, it refers to a known referent: in the program, the constant symbols are the same symbols that are used for elements of the universe. The symbol, used as a formal language symbol, always refers to itself as a universe element. This is, a notational and implementational convenience.

The function of no arguments corresponds to an attributive noun phrase, it refers to some unknown referent. This referent must be one of the elements of the universe, but exactly which one is unknown until a model has been successfully built.

This distinction could be helpful in implementing a natural language interface to the program. This point is related to sections 4.1 and 4.6 and will be included in a later memo discussing natural language interfaces to sorted logic.

2.1.4 Non-standard features

Finally, in this section, brief mention of two unusual features. Firstly, the formalism allows use of a "partitioning" connective. Hayes [Hayes, 1979] calls formulae resulting from this connective "taxonomies"

by analogy with classification. I shall refer to such formulae as "partitionings". The partitioning connective (like most of the connectives used in this system) is n-ary. As an example:

P ‖ Q ‖ R ‖ S

is a partitioning of P, Q, R and S, where P, Q, R and S are boolean valued formulae.

The meaning of this example formula is that exactly one of the four sub-formulae holds (is true). If none of them are true, then the partitioning has the value "false". Thus we could have the following "taxonomy":

```
(x).   VertebrateEx]
   <=> (   MammaIEx]
         ‖ BirdCxD
         ‖ ReptileCx]
         ‖ FishCx]
         ‖ AmphibianCxD)
```

No value is defined when more than one of the sub-formulae is true, so such a partitioning would be ill-formed.

The other non-standard feature is the use of "generalised quantifiers". These are a generalisation of the usual logical quantifiers, again to increase the expressive power of the language. They will not be discussed further in this paper, for more information see [Cunningham, 1984].

Both the above features are reducible to equivalent first order expressions with the usual connectives and quantifiers.

2.2 Control

The purpose of the program is to accept a collection of axioms, and to generate a database representing a possible model for the axioms. The program takes as input sort structure specifications for the universe, sorting functions (see section 1.2) for the predicate and function symbols of the language, a collection of axioms and, optionally, some initial fragment of the model. Then the program works by checking that axioms evaluate to "true". Since a given atomic expression may have different values in different models, it is not possible to infer every value in a model. So the program assumes values for some atomic expressions, when they are not otherwise constrained. It doesn't assume a value for some expression if the value of the whole expression can be determined without it. For example, if, evaluating the expression:

P and Q and R and S

the program discovers that, say, R evaluates to "false" it is unnecessary to evaluate, or assume values for, any of the other sub-expressions.

The checking process is necessarily fairly simple (for efficiency reasons), so this assumption of values can occur even when, theoretically, the value is determined by the requirement for a consistent model. Consequently, it is possible to have two contradictory assumptions. In these circumstances, the program starts reasoning at a "meta-level" about the various assumptions it has made, in order to decide which of them to reject. When it rejects an assumption, a form of "truth maintenance" is necessary [Doyle, 1978], because the value of other formulae in the system will be contingent upon the rejected assumption.

3. Requirements for an Expert System

The task an expert system must perform can be seen as very similar to the task this model building program must perform. Here I define the requirements of an expert system:

3.1

It must accept general domain specific knowledge in a form conveniently specified by a human expert. In many earlier systems this knowledge is effectively in the form of "IF ... THEN ..." rules, that is, rules with some condition and a consequent action or inference, although these rules may be organised into a network.

3.2

It must be able to accept volunteered information about a particular problem within that domain. This volunteered information should be acceptable at any time in the "expert" consultation.

3.3

It must ask questions when it would benefit from being given the answer, but it should also be able to continue if no answer is available. Also, it should be able to accept an indirect answer. By this I mean that the user of the system should be able to volunteer information in answer to a question (as specified in point 3.2 above), and if this information makes it unnecessary to obtain a direct answer, it should not re-ask the original question.

3.4

3.4.1

Answers to questions and volunteered information may be uncertain or incomplete. An expert system should allow this, and be able to make sensible use of such information.

3.4.2

The human expert's knowledge may be heuristic and even inconsistent. In particular, the use of defaults should be permitted. An example default rule is "All people have two legs[11], which could be disregarded in cases which were exceptions. An expert system with such a default rule should accept a statement like "Long John Silver has only one leg".

3.5

An expert system should be able to explain its reasoning. By this, I mean two things. Firstly, it should be able to provide a justification of any conclusions it reaches (either intermediate or final). Secondly, it should be able to describe what inferences can be made from a particular piece of knowledge, and hence what use might be made of the answer to some question the system has asked.

Both these kinds of explanation should be in a form comprehensible to a human, although it is acceptable that the human has to be expert in the application domain in order to understand the explanation.

3.6

This is a related requirement. When the system asks questions there should appear to be some purpose to them. The system should not appear to be asking unrelated questions. This is clearly an important

requirement when the system is being used by a human expert (and hence when a system is being developed), but I am not sure to what extent this requirement would be necessary if the system were being used by someone unfamiliar with the problem domain. (Would a person unfamiliar with a given problem domain be able to understand the point of questions a human expert would regard as relevant? If not, they couldn't be concerned by apparently irrelevant questions, since for them all the questions would seem obscure.)

3.7

The system should be able to work in conjunction with a human expert, so that at one stage the system might take the initiative but a human expert could advise the system that it was working on an incorrect hypothesis, and the human could suggest an alternative strategy.

4- Using the Model Builder

I shall consider each of these points in relation to using my existing model-building program as an expert system.

4.1

The particular sorted logic used here is a convenient formalism in which to express expert rules. Any statement expressible in the full first order predicate calculus is expressible, so the system can accept information in a more convenient form than most current rule-based expert systems. In particular, the "generalised quantifiers", referred to in section 2.1.4, allow easy expression of some classes of statement which are normally awkward to represent in a predicate calculus. It is not, of course, immediately suitable for use by someone unfamiliar with

formal logic. However, it is a tractable research problem to translate highly domain specific natural language into this sorted logic and the task is made easier by the interactive nature of an expert system. Obviously this statement could be contentious, and I intend to make it the subject of a later memo. There is a related problem here with natural language output - see section 4.6 below.

4.2

Since the program works by building a model satisfying some initial formulae, there is no difficulty in adding volunteered information to the model before building commences. Once the program is in operation it may make assumptions, either using defaults or as working hypotheses. Volunteered information can conflict with such assumptions, but this is no harder to deal with than conflicts between internally inferred data and assumptions. The existing program mechanisms can cope unchanged.

4.3

Only a primitive form of question asking has been provided in the existing program.

4.4

4.4.1

The program already distinguishes between known facts and assumptions, thus it already distinguishes between certain and uncertain knowledge. It may be desirable to extend the range of possible "certainties" with which the program works. This could, if desired, be done numerically by associating numerical "confidence" values with the inferred and hypothesised data. A more interesting approach lies in using the

system's meta-reasoning capabilities. This is discussed further in section 4.6 and the section on control strategies (5.2) below.

4.4.2

The program distinguishes between two kinds of knowledge. The first kind, corresponding to expert domain knowledge, consists of 'axioms'. These are well formed formulae in the full sorted logic. (The program also needs to be given information about the sort-structure assumed by these axioms.) The other kind of knowledge corresponds to 'facts' about a particular problem within the domain. For the program, this would be a specification of part of the mapping from atomic propositions and terms to the model. At present, the program only allows uncertainty in this second kind of knowledge, and it requires definite 'axioms'. The modification to the program to enable it to use default axioms, and uncertain axioms, is again fairly straightforward.

There is a real distinction between "uncertain" axioms and "default" axioms. The difference lies in the effect of discovering some fact (or assumption) that contradicts the axiom. As an example, consider the axiom "all birds can fly":

(x).Bird[x] => Can_fly[x]

Suppose the universe contains three birds: an owl an eagle and an ostrich. What should be the effect of discovering that the ostrich cannot fly?

If the axiom is (to be used as) a default, then we simply dis-believe that the ostrich can fly. It is as if the axiom is a concise representation of three separate assumptions: that the owl can fly, the eagle can fly and the ostrich can fly. The assumptions are unconnected,

so discovering that one is wrong has no effect on the others.

If the axiom is uncertain, then discovering one exception may cause complete loss of confidence in any inference from the axiom. Suppose we think it possible that "all pigs have wings[11]:

(x)PigCx3 => Has_wingsCx]

We then encounter Porky the pig, and discover him to be wingless. This could shake our belief in the wingedness of the pigs Pinky and Perky.

When the axiom is quantified over more than one variable, the possibilities get more complex. As a further complication, we may wish to have a spectrum of possibilities between the "default[11] kind of axiom, and the "uncertain[11] kind of axiom. Alternatively, the "default[11] and "uncertain" properties can be considered as distinct notions so that, for example, if we know nothing about whether birds fly, and the first bird we discover is an ostrich, we may postulate an uncertain default axiom:

(x).BirdCxD => not CanJlyCxD

The notions of uncertain and default axioms need further consideration.

4.5

In order for the program to have a form of truth-maintenance system, it must record reasons for all of its beliefs. Thus, in order to justify any given assertion to a human user, it is able to refer to the rules, data and hypotheses on which that assertion is based. Ideally, a natural language description of, say, a rule would be generated from the actual sorted logic formula representing that rule. This would be in contrast to most existing expert system technology which requires an "explanation

text" to be associated with each rule, although there are exceptions.

4.6

Making the system ask apparently purposeful questions is perhaps the most challenging part of the task of modifying the existing program. At present the program has a very simple control strategy, and although a simple question asking modification has been added (mentioned in 4.3 above), that will ask a group of related questions for one rule, there is no control over how it chooses rules to consider. However, this control problem has been considered in the design of the program. When the program discovers an inconsistency in its hypotheses, it starts reasoning at a meta-level (using the same mechanisms as for the original problem). At this level it is reasoning about the assumptions and symbols used for the original problem. It is not until a consistent meta-model has been found for the original problem that the original problem can be resumed. Thus the making of an assumption is determined by procedures working in the problem domain but these procedures can access the meta-level database to determine how to make an assumption. An assumption can only be rejected after doing some meta-level reasoning about that assumption and its relationship with other assumptions.

The natural way to handle control questions in this framework would be for the control decision procedures to be influenced by the facts and hypotheses about the non-logical symbols of the problem domain which are present in the meta-level database. Since the mechanisms used in the meta-domain are the same as those used in the problem domain (implying the existence of a meta-meta-domain), there could be interaction with the user about the meta-domain. Also, the program would be capable of making assumptions <u>about</u> axioms from the problem domain. It might even

conclude that some default rule should never be used! Such meta-information, being about general rules and defaults, could be allowed to persist from one specific problem in the domain to another: thus providing a limited learning capability. These issues are still being investigated, but there is some discussion in section 5.2.

4.7

The requirement of section 3.7, for interactive operation and control guidance, is naturally provided by the strategy discussed in section 4.6. The meta-domain is concerned with the relationships between assumptions, rules and facts, and with issues of control. It works in exactly the same fashion as the original problem domain, so requirement 3.7 can be reduced to requirement 3.2 at the meta-level. Notice also that every time a difficulty is encountered with the inference process at the problem level (such as a contradiction between assumptions), a consistent resolution of this must be found at the meta-level. This, as well as resolving the conflict, might very easily result in a different consistent model of the best control strategy for the problem domain. If the system described this consistent control strategy, it would be keeping the user informed of just what it was trying to do - in fact, the user could have been participating in the formation of the control strategy.

5. Discussion

In this section I return to those points, above, which needed further exposition.

5-1 Computational advantages of Sorted Logic

This section only discusses the computational advantages of sorted logic, although there are other advantages for a human user. There are three main computational advantages:

5.1.1 Conciseness

Since shorter expressions may be used in the sorted logic than if the extra information derived from the sort structure had to be added explicitly, there is a computational saving simply because the knowledge represented takes less space. Also, it is possible to "traverse" through the sub-formulae of a small formula more quickly than through a larger formula, so there *are* time savings as well. (This assumes some sensible representation of the sort information.)

5.1.2 Separating the sort information

Sort information is supplied to the program separately from the sorted axioms. This is an advantage for two reasons.

Firstly, it is easy and natural to impose a sort structure on a problem domain. When this is not done explicitly (as in an unsorted logic), the knowledge this embodies must still be present but is scattered amongst the other pieces of knowledge. The separation of the sort structure enables the automated checking of the consistency of the sort structure. It does not seem necessary for this part of a problem to be specified with uncertainty or defaults.

Secondly, since the sort information relates to the whole problem domain, rather than to individual problems in the domain, this automated checking need only be done once. If the sort information were mixed in with other information, it would effectively be checked as part of the model-building each time the system tackled a new problem.

5.1.3

Unlike theorem proving, building a model may involve checking the values of formulae with many different substitutions of universe elements for variables. Suppose, in a simple problem, there were five disjoint sorts, each with four universe elements. Then an axiom universally quantified over three variables could have (4 x 4 x 4 = 64) possible substitutions. In principle, it seems that it may be necessary to check that the axiom holds (is true for) each of these sixty four substitutions.

The equivalent unsorted axiom, as well as being longer, would still have three universally quantified variables, but each of these could now range over the whole universe, giving (20 x 20 x 20 = 8000) possible substitutions. Even though most of these would be trivially true (because they would correspond to inappropriate substitutions of variables), it would still be computationally expensive.

5.2 Control Strategies

The program works by attempting to ensure that all the axioms it is given evaluate to "true". The control problem is to determine what order it does things in order to achieve this. At present, only a limited number of simple heuristics are in use, but the system has existing mechanisms to enable it to use "meta-reasoning" to guide a control strategy.

There are two constraints on the control strategy. The first constraint is that the program should be reasonably efficient. The second constraint is that described in section 3.7: that the system should appear to ask sensible questions.

5.2.1 Efficiency

There is only one important requirement necessary to achieve efficiency: the program should not make incorrect assumptions.

To avoid incorrect assumptions, the program can partly rely on heuristics, but must mainly rely on not making any unnecessary assumptions. For this reason, the program works in one of four "modes" of operation, switching between them as necessary. The modes are called "S", "U", "D" and "A". In modes "D" and "U" the program does not make any assumptions, and only in modes "D" and "S" does the program make inferences. Mode "D" is the preferred mode of operation.

The main efficiency gains could be achieved by evaluating axioms in some optimal order, and by changing modes optimally. (If it is necessary to know a value for some term but the actual value is unimportant, then it is better to assume some value, which can only be done in modes "S" or "A", than to continue to attempt to infer the value in mode "D".)

5.2.2 Sensible Questions

I conjecture that a good control strategy, if determined using the meta-reasoning capability mentioned in section 4.7, would also be "sensible" in the sense that it would tend to discover contradictions between assumptions as soon as possible, and hence to ask questions which were related. To do this well might require the ability to switch from evaluation of one axiom to evaluation of another axiom which had terms in common with the first axiom. At present, the program can perform this switch of axioms after doing some meta-reasoning. However, there is no clever method of determining which axiom to consider next: only simple heuristics are used at present.

The system is designed to allow its control strategy to be influenced by

its meta-level reasoning.

5,2,3 Meta-level Reasoning

As implied in section 4.7, the system can switch to a meta-level copy of all its data-structures, in order to reason about the problem domain as opposed to reasoning in the problem domain. Certain procedures, most notably those controlling the making of assumptions, can access the meta-level in order either to add information (e.g. about a new assumption), or to inspect data-structures (e.g. to guide what assumptions to make).

At present, this mechanism is not used by the control structures, but there would be no difficulty in using the existing mechanisms for this purpose.

5.2.4

This meta-level reasoning capability seems likely to be most useful when more realistic expert domains are investigated, which involve different kinds of uncertain knowledge. For example, it may be useful to use "default[10] axioms before making any assumptions about the values of terms occurring in such axioms, but perhaps "uncertain[11] axioms should only be used after all other axioms have been considered.

There is also scope for generalising the notion of "model[11]: for example, finding partial models using only modes "0" and "U" (i.e. no assumptions). There could be different kinds of axiom: those which are completely satisfied by a model, requiring evaluation in all four modes; and those only used to restrict the model for the first kind of axiom, and evaluated only in modes "D[11] and "U[11].

The exact status of any axiom would be represented as a meta-level

assertion. Since the meta-level works in the same way as the problem level, this representation strategy allows for the status of axioms to be left undefined, and hence assumed by the program.

6. Conclusion

It has been shown that there are definite similarities between the inference part of the task that an expert system must perform, and the task of constructing a model satisfying a collection of logical formulae. Whether or not the greater generality of the model building task must impose too great a barrier to efficiency of the system when used as an expert system is an open research question, although the lines of research described in section 5.2 would suggest that it is not.

There are definite advantages to be gained from using formal logic as the basic representation language of an expert system. These lie chiefly in the well-defined semantics for formal logic, which would allow clean interfaces between the inference component of an expert system, and other parts of the system: the natural language interfaces and the control component. Other systems (such as automatic planning, or theorem provers) could also be cleanly interfaced. This paper has also assumed that translation between sorted logic and natural language is relatively tractable, although it was beyond the scope of the paper to present the supporting arguments.

Use of logic in the control component of the system allows the elimination of the ad-hoc numerical computations at present performed by most expert systems. Again, the well-defined semantics allows clear expression of exactly how uncertain knowledge is being handled by the system. This extra clarity is illustrated by the distinction made in

section 4.4.2.

In conclusion, there are a number of substantial issues which are worth tackling. It would be worthwhile to attempt to build an expert system along the lines described in this paper.

References.

Cohn, A G, (1983), "Mechanising a Particularly Expressive Many Sorted Logic", PhD Thesis, University of Essex.

Cunningham, JL, (1984), "Generalising Quantifiers", Cognitive Science Research Paper #30, University of Sussex.

Donnellan, Keith (1966), "Reference and Definite Descriptions", Philosophical Review LXXV, pp 281-304.

Doyle, J, (1978), "Truth Maintenance Systems for Problem Solving", MIT Artificial Intelligence Laboratory Report 419, January 1978

Hayes, P J, (1971), "A Logic of Actions", Machine Intelligence 6, Edinburgh University Press.

Hayes, P J, (1979), "The Naive Physics Manifesto", Expert Systems in the Micro Electronic Age (ed D Michie), Edinburgh University Press.

Tapscott, B L, (1976), "Elementary Applied Symbolic Logic", Prentice-Hall