AN ENVIRONMENT FOR EXTENDING CONVENTIONAL PROGRAMMING LANGUAGES TO BUILD EXPERT SYSTEM APPLICATIONS

F. Franek[1] and I. Bruha[2]

Dept. of Comp. Sci. and Systems
McMaster University
Hamilton, Ontario
L8S 4K1 Canada

[1] Research supported by NSERC OGP0025112 research grant.
[2] Research supported by NSERC A20037 research grant.

ABSTRACT

McESE (McMaster Expert System Environment) is a software tool for building problem-specific shells and creating expert system applications in a particular programming language. Developed and implemented by the authors at McMaster University, it is designed to allow the user to deal with imprecise and incomplete knowledge, customize the shell's handling of uncertainty, allow hierarchical partitioning of knowledge bases, and fast prototyping. Specialized software of McESE is written in C and facilitates handling of all aspects of dealing with rule-based knowledge bases. Some of practical and theoretical aspects of McESE are discussed in this paper. For more details see [FB], [FB1], [F], [L].

INTRODUCTION

McESE is an expert system environment designed to build problem-specific shells and create expert system applications. It is designed to satisfy the following goals:

- allow the user to deal with imprecise and incomplete knowledge in McESE knowledge bases with a declarative formalism that has a satisfactory degree of expressive power;
- allow the user to customize the shell as so it handles uncertainty in the way of his preference;
- allow the user to create expert system applications in a particular programming language (C, FranzLISP, and SCHEME are available at the moment), with a point of reference being the application rather than the knowledge base (so the creation of such an application resembles ordinary programming as much as possible);
- allow the user a natural (hierarchical) connection of different knowledge bases in an application;
- allow rapid prototyping;
- allow fast inferring.

DESCRIPTION OF McESE

In McESE the user can encode the domain knowledge in general rules of the form:

TERM1 & TERM2 & ... & TERMn ==CVPF==> TERM

where CVPF abbreviates "certainty value propagation function". The "meaning" of a simple rule ***TERM1 & TERM2 ==F==> TERM3*** is: if we are certain with value ***v1*** that ***TERM1*** is true, and if we are certain with value ***v2*** that ***TERM2*** is true, then we are certain with value *F(v1,v2)* that ***TERM3*** holds.

A term has the form:

weight * predicate_name (list_of_variables) threshold_directive

where ***weight*** and ***threshold_directive*** are optional, ***threshold_directive*** having form **[*top tval*]**, where ***top*** is > or >=, and ***tval*** a real value between 0 and 1 inclusive. The predicate in the term may be negated by the symbol '-' or '~'.

Firing of such rule (after all variables have been bound to objects) consists of the evaluation of the right hand side (RHS for short) predicate using certainty values of all left hand side (LHS for short) predicates (and hence terms) processed by the corresponding CVPF.

Rules in this form allow to capture an imprecise, uncertain and incomplete knowledge, since the rules are guaranteed to "fire" for any values of LHS terms (except some situations when the firing is prevented by the CVPF), and only the resulting value of the RHS predicate is affected by the values of LHS terms.

The predicates serve as simple statements about relations among objects of the domain the user is "dealing" with.

For a particular knowledge base, if no CVPF in a rule is stipulated, the default one is used. As any CVPF can be defined as default by the user, he can in fact pre-determine that all rules in the knowledge base will be handled uniformly, in essence fixing a particular method of the treatment of uncertainty for the knowledge base.

Most of expert systems shells are either presented with the knowledge representation language as the main language of the application, and hence the application is "centered" around the "model" (knowledge base), and the procedural parts are connected to it by different means (in the case of OPS languages and PROLOG it is the only language), or they themselves are written in the language of application (for example KEE in LISP). We tried to give the user a possibility to write an application in the usual way, at least the procedural parts, and in the programing language of his choice, but still preserve the possibility of having access to a declarative knowledge base when needed. This is achieved by "extending" a particular programming language by McESE commands to facilitate all required communication between the application and the knowledge bases. The software to perform the communication is written in C, but is transparent to the user. Thus, a particular application is completely built using a single programming language and the language of McESE rules. At this point, McESE extensions of C, FranzLISP, and SCHEME are available. Note that this shift in emphasis changes the focal point from knowledge base to the application in an effort to allow for ordinary programming techniques, methods, and experience to be fully utilized.

Predicates which never occur on RHS of any rule correspond to facts and observations; we shall call them level 0 predicates. They represent data input nodes of the knowledge base. Their values are not derived (inferred) using rules, they must be obtained from so-called predicate service procedures. These may be ordinary procedures to supply the facts and/or observations, or they may in fact be other expert systems. This mechanism allows for convenient partitioning of the domain knowledge into a hierarchy of knowledge bases (or more precisely expert systems), see Fig. 1.

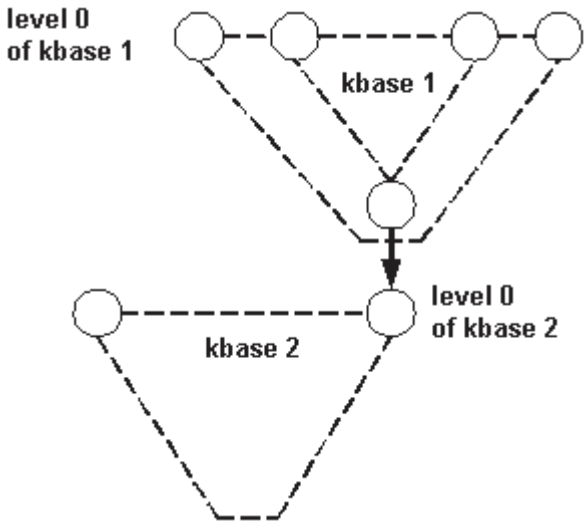

Fig. 1

On the other hand, predicates on higher levels represent conclusions based on facts and/or other conclusions. Their value must be obtained by inferring.

Since McESE built-in inference engine automatically prompts the user for the result of the invocation of a predicate service procedure in the case that the predicate service procedure is not available to the system (and similarly for CVPF's), one can just test and modify rules in the knowledge base without the overhead of building the complete application. Moreover since McESE interactions and inferences are identical in McESE-C, McESE-FranzLISP, and McESE-SCHEME, one can quickly build a prototype in McESE-FranzLISP (utilizing versatility and flexibility of FranzLISP) to verify the methods and approaches, and when satisfied, the knowledge bases can be used as they are for the McESE-C application.

McESE knowledge bases are compiled immediately while being created and/or modified. The resulting data structure allows for direct linking of relevant predicates, so only relevant rules are considered when a predicate is being evaluated. Thus inferring with such a knowledge base amounts to a "walk" through the data structure and hence the speed of inferring depends entirely on the depth of the knowledge base rather than on its size. The result is a fast performance, knowledge base queries are quickly evaluated and returned to the application program.

McESE editor allows the user to edit a knowledge base (i.e. the rules) in the declarative form though the knowledge base is maintained as the above described data structure. The editor provides on-line syntactical checking, parsing and compiling of the rules, and a rudimentary semantical checking. This greatly simplifies the task of creation and modification of knowledge bases.

McESE inference engine provides the mechanism for "inferring". It can work in two basic modes, forward chaining and backward chaining. Backward chaining from a given predicate with given bindings for its variables is performed as depth-first walk down the (compiled) knowledge base to level 0 predicates (with simultaneous propagation of bindings for predicate variables), "selecting" the appropriate substructure leading to the predicate being evaluated (there may be more than one such substructure). Only the required 0 level predicates are activated (and so appropriate known facts are fetched and/or appropriate observations are made) and then the resulting certainty values are propagated (and recorded in the knowledge base, too) back through the selected substructure to the required predicate. The backward chaining mode has four submodes which amount to rule conflict resolution: max mode, sufficient max mode, min mode, and sufficient min mode. Sufficient max (min) mode searches for a substructure which evaluates the required predicate to a certainty value bigger (less) or equal to the specified value, then the chaining stops and this value is returned, or if such a substructure is not found, "failure" is returned. The max (min) mode evaluates all possible substructures and returns the max, i.e. the highest value (the min, i.e. the lowest value respectively). Forward chaining is implemented only from 0 level up, to a specified level. Specified predicates from level 0 and their ascendants up to the specified level are evaluated and their values recorded in the knowledge base for latter use.

The inference engine can work in two modes as far as explaining what it is doing: the silent mode when all inferring is transparent to the user and only the resulting value is available, or in trace mode when all inferring is done on the screen, rule by rule, predicate by predicate, with all relevant information being displayed, too. The trace mode is useful mainly for testing and debugging of knowledge bases.

A run time consistency checking takes place: only the minimal and maximal values for a predicate are recorded. The difference between these two is the inconsistency level. McESE allows the user to preset for a knowledge base what inconsistency level it can tolerate. If the inconsistency level tolerance is exceeded, ALARM is issued and the inferred value is returned to the application.

Also, a run time completness checking takes place: in the case a predicate cannot be evaluated, ALARM is also issued and "failure" is returned by the inference engine.

For each McESE knowledge base the user can specify ALARM procedure which is automatically invoked by McESE when ALARM is issued. In this procedure the user can define what should be done.

An invocation of the inference engine from the application constitutes one inference cycle. Any of the values recorded in the knowledge base as results of backward or forward chaining within the last inference cycle can be fetched to the application, if needed.

As a simple explanation mechanism the inference engine keeps track of the substructure used for the max evaluation for each predicate evaluated during the last inference cycle (and similarly for the min evaluation), and displays it when asked for, together with the input data which affected the particular values of facts on level 0 at the time of the evaluation. Thus, the user can trace back any inference to the facts and observations of level 0.

SOME THEORETICAL ASPECTS OF McESE.

"Deep knowledge" versus "shallow knowledge"

Expert systems of the so-called first generation deal with what is called "shallow knowledge", i.e. the knowledge being just a set of heuristics, without any "deep" understanding of the domain, where all concepts are treated uniformly in a homogeneous way. On the other hand, so-called "deep knowledge" calls for explicit models of the domain embodying "understanding" of different concepts within the domain. There have been quite strong claims with regard to expert systems of the second generation using this "deep knowledge" (see e.g. [NSM]), but we tend to agree with [S] that these are mostly unsupported by substantial practical demonstrations yet. Being guided by our pragmatical goals, McESE belongs among first generation expert systems, with all the consequences for explanations and knowledge acquisition.

Knowledge acquisition in McESE

As stated many many times, knowledge acquisition is the real bottleneck of expert systems development. It is very hard to get and formalize domain knowledge, and the task is time consuming and seemingly endless. Besides that, the domain knowledge may (and will) differ from one human expert to another. There are hopes (and claims) that the use of "deep knowledge" expert systems will alleviate this problem and will help facilitate in some way "learning" (see [S]). Given the current state of the research into these problems, McESE does not address it in any way and leaves it to the poor user (i.e. knowledge engineer) to deal with it in the old fashion way. There are, though, some interesting possibilities for "learning by observation" the CVPF's (or their modeling by neural nets), if given the concepts and their relations. This is in the plans for our future research concerning McESE.

Treatment of uncertainty in McESE

Presently there are two basic approaches to uncertainty in expert systems.

The first, linguistic approach, is what one may describe as "reasoning about uncertainty", where linguistic terms are used to deal with uncertain aspects of the knowledge (as

we humans do), and their interpretation and verification is handled according to the "context" in which they were posed. The argument of people following this line of thought is a plausible one, namely that the knowledge representation we use ought to reflect the way we communicate our ideas as much as possible. But, from the pragmatical (computational) point of view this is not (at least at this stage) a very practical approach due to the need of a very sophisticated mechanism to handle this interpretation and verification of linguistic terms of uncertainty. We suspect that this approach may be feasible when systems for representation and reasoning with common sense are developed. For example see [CG] for a heuristic treatment of this type. There are other schemes in this category, like default reasoning, but we think they address more the problem of reasoning with incomplete and/or unreliable knowledge.

The second approach is a numerical one, where numbers are employed to indicate the degree of certainty, or uncertainty, or whatever other term may be used. This seems to be more natural (at least at this stage) and the field is richer in different schemes how to do it. We are inclined to view the problem of uncertainty in this light, given the fact that even we, humans, very often have to use numerical specifications to clear some aspects of our transmission of ideas. When we were deciding how to deal with uncertainty in McESE, we were mainly guided by admittedly pragmatical aspects of the proposed system. Thus the question was which of many schemes to employ. For an excellent overview of the field, see [G]. Briefly, there are about four main ways to deal with uncertainty numerically.

Probabilistic (Bayesian statistics), which has the advantage of a well developed terminology, technology and machinery, and also is easily comprehensible to humans as we have incorporated probability into our everyday lives and language. Some of the works in this field clearly influenced our thinking, especially for their clear "theoretical" approach and quite impressive theories (see e.g. [DP], [P], [RP]). But given the fact that one has to establish a host of prior probabilities in the "background" to facilitate the complete probability distribution and/or assume that most of "events" are statistically independent, renders the whole approach computationally almost impossible, or one must fit (and that is the most frequent case) his knowledge representation into a fixed scheme satisfying the underlying (and often implicit and hence "transparent") restrictions. These systems have what we call one degree of ad-hocness in that that the prior probabilities of the "events" are ad-hoc (i.e. supplied by the domain expert), their coherent processing is well-established and well understood process, and so easily interpreted (some mathematicians argue though that the coherence is only a superficial one, that the approach is meaningless as the "events" in question are not really part of a chance system).

Evidential approach, mostly based on variations of what is now commonly called Dempster-Shafer theory of belief functions, presupposes that the pieces of evidence are "independent", so again one has to mold his knowledge representation into a fixed scheme satisfying some (unfortunately implicit and not so obvious) underlying restrictions. Similarly, as with the probabilistic approach, the penalty to pay is that the system becomes incoherent (theoretically, it does not have to be visible on the performance of the system in question) when these underlying restrictions are violated, with no recourse but re-fitting the knowledge representation into the required framework. Since these schemes are computationally even more taxing than the probabilistic approach, applications in the field tend to simplify matters by

considering only the simplest of belief (support) functions. These systems have what we call two degrees of ad-hocness, for these evidential support belief functions must be supplied for each piece of evidence to be considered in order to get a meaningful interaction among all "events". They are propagated using some (slightly ad-hoc) machinery of combining evidence (Dempster's rule), and then the results must be interpreted (what they mean). The violation of underlying constrains is even less obvious than in the probabilistic case. On the other hand, there are common grounds between those two, as in the case of causal trees (see [SS]).

The third approach is "fuzzy logic" or "fuzzy set" approach. There every linguistic (uncertain) term is represented by a membership function and the degree of uncertainty reflects in the membership function of the "conclusion". There are some common grounds with Dempster-Shafer theory (see [Z],[Z1]), but we have not seen a practical application yet going beyond the use of max and min operators to propagate "certainties" through rules (corresponding to conjunction and disjunction respectively). These systems have what we call three degrees of ad-hocness, for one has to supply completely ad-hoc membership functions for each linguistic term, these are processed with an ad-hoc mechanism to obtain some results, whose interpretations are again ad-hoc. More than in the previous two approaches, the underlying constrains (with respect to max and min operators) are so tied, that the system becomes factually incoherent quite fast.

The fourth way to deal with uncertainty may be called ad-hoc systems. There belong systems with different mechanisms of propagating "certainty factors" (MYCIN - see [BS]), and so. The list would be quite long. These systems exhibit three degrees of ad-hocness (ad-hoc numbers to start with, ad-hoc propagation mechanism, and ad-hoc interpretation). However, they proved themselves quite well from the practical point of view.

In the light of the above mentioned possibilities, bearing on our mind one of our most important goals (speed of execution, and thus a need of a simple mechanism) we opted for a "hybrid" solution exhibiting also three degrees of ad-hocness. The certainty values passed to the knowledge base are ad-hoc in that that they are results of "computations" of level 0 predicate procedures and reflect the user's ideas about how certain he is about them. The CVPF's are ad-hoc, for it is up to the user to provide them, according to his "feelings". And finally the interpretation of the resulting values is also ad-hoc, for their is no "theoretical" explanation of what these numbers mean. Why did we opt for a system with three degrees of ad-hocness? Speed itself would not justify it, as having for example one or two functions to propagate the certainty values would be even faster and more convenient for the user. (a) we had found systems of all four numerical approaches inherently inflexible. In brief, you have to "mold" the rules to fit the system, you have no freedom to capture the "structural" relationships of the predicates involved, and then work on capturing the "certainty value" characteristic of the rule. Our approach allows you to do it in two separate steps. (b) we had found that too often we needed different approaches to uncertainty within a single knowledge base (and/or control mechanism). Again, CVPF's allow you to "switch" from "probabilistic" to "evidential" to "fuzzy" to "ad-hoc" on the go. (c) we had been influenced by (very pointed) arguments by proponents of non-numerical approach, especially "heuristic" approach. Again, CVPF's allow us to do emulate it to a degree. They are nothing else, but (procedural) heuristics about

uncertainty in the particular rule. (d) we had found that very often we were able to formulate the "structure" of a rule correctly, while the numbers needed frequent corrections (tuning). The separation of the structure of the rule from the interpretation of uncertainty in the rule helped to solve this. If the structure of the rule is satisfactory, one need to fine-tune the CVPF's only. (e) there has been an additional bonus in the possibility to run some "experiments" with knowledge bases under different interpretations of uncertainty.

CONCLUSION

McESE in its initial phases and forms (McESE-FranzLISP, McESE-C, and McESE-Scheme) has satisfied our expectations. It has met the goals stated when we started the project. It serves as a test bed for different approaches to uncertainty for our research, as well as an expert system environment for our Expert Systems courses (especially McESE-FranzLISP).

REFERENCES

[BS] B. Buchanan, E.H. Shortliffe, Rule-based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Addison-Wesley, 1984.

[CG] P. R. Cohen, M. R.Grinberg, A Framework for Heuristic Reasoning About Uncertainty, IJCAI 85 Proceedings.

[DP] D. Dubois, H. Prade, Combination and Propagation of Uncertainty with Belief Functions, IJCAI 85 Proceedings.

[G] Artificial Intelligence and Statistics, edited by W. A. Gale, Addison-Wesley, 1986.

[NSM] R. Neches, W. R. Swartout, J. Moore, Explainable (and Maintainable) Expert Systems, IJCAI 85 Proceedings.

[P] J. Pearl, Fusion, Propagation, and Structuring in Belief Networks, Artificial Intelligence 29 (1986).

[RP] I. Razier, J. Pearl, Learning Link-Probabilities in Trees, Proceedings of the Workshop on Uncertainty in Artificial Intelligence, University of Pennsylvania, 1986.

[S] D. Sharman, A Review of Recent Developments Relating to Deep Knowledge expert Systems, research Report No. 86/236/10, University of Calgary, 1986.

[SS] P. P. Shenoy, G. Shafer, Propagating Belief Functions Local Computations, IEEE Expert, Fall 86.

[Z] L. A. Zadeh, The Role of Fuzzy Logic in the Management Uncertainty in Expert Systems, in Fuzzy Sets and Systems, North-Holland, 1983.

[Z1] L. A. Zadeh, Syllogic Reasoning as a Basis for Combination of Evidence in Expert Systems, IJCAI 1985 Proceedings.