DEVELOPING EXPERT SYSTEMS BUILDERS IN LOGIC PROGRAMMING

Eugenio Oliveira
Dept. Informatica,
Universidade Nova de Lisboa
Quinta da Torre
2825, Monte da Caparica

ABSTRACT

We intend to develop a set of kits to build Expert Systems using Prolog. Two principal modules, a Knowledge Base acquisition and consultation subsystems are now presented.

Several knowledge representation structures and mixed inference mechanismes are proposed for the sake of system efficiency. Finally some explanation capabilities derived accordingly with used inference methods are also implemented and presented.

.Introduction

Knowledge Based Systems are typical, useful and practical Artificial Intelligence applications.

Knowledge Representation schemas, Problem Solving methods, Natural Language interfaces, Knowledge acquisition capabilities, Plausible reasoning are several important techniques we can find inside AI to build up more intelligent systems to perform expert's knowledge into a great variety of domains.

Knowledge is the very fundamental component of such systems. Nevertheless, if such systems may obey the paradigm of being " Knowledge rich even if they are methods poor ", efficiency and friendliness must not be neglected for the sake of usefulness.

Our experience with Expert Systems (ES) -- Knowledge Based Systems embodying knowledge of one or more experts in a given domain (medicine, geology, ecology, business...) -- gave us some particular insights in such a tradeoff. So, a number of design ideas we now present evolved from past work in ORBI.

ORBI [PER] is an Expert System designed for environmental

resource evaluation, written in PROLOG and running on a PDP 11/23 which gives advice about regions aptitudes and resources. It has a dynamic Knowledge Base entered and modified by experts (not programmers) and supports its decisions with more or less detailed explanations about its reasoning.

One of the fundamental lessons of ORBI development and implementation is PROLOG suitability to encode in a declarative manner structured knowledge about the world (semantic networks, production rules...) as well as the query language, relational database, intermediate interpreters, all of this with the same clear formalism (Horn clause logic).

One of the important drawbacks of most existing systems is that they reflect specific domain particularities loosing all the generality.

Other critical point of such systems is the difficult acquisition of new knowledge and modification of old one directly from experts without the need for computer scientists.

Recent developments show some attempts at generalizing pre-existing ES, trying to present domain independent mechanisms and to be a general framework to deal with at least some classes of worlds.

It is our aim to develop more versatile, powerful and simple Expert Systems Builders using Logic Programming.

.System organization

Our system is able to acquire interactively all the concepts of each new world, to represent them internaly, to relate them, to display them in a comprehensive manner on user's demand. It must have an efficient and versatile procedural behaviour to achieve intended results well enough supported with explanations.

The system can be regarded as two main cooperating modules :

--- Knowledge Base Acquisition Subsystem (KBAS)
--- Consultation Subsystem (CS)

KBAS guides the expert accepting his structured knowledge, individualizes and defines domain concepts, keeps all existent relationships, so entering a complete new world into the system.

.Knowledge Base

Each entity is a triple <concept, attribute, value>. With these entities a conceptual semantic network is built up,whose nodes, corresponding to single concepts (for example "disease"), are expanded on records with several fields (for example meaning, number of attributes...). One of these fields is pointing another tree of concept's attributes each of which with its own characteristics.

We can see this part of Knowledge Base as organized into three layers :

--- Templates, abstracted schemas for concepts's characteristics and rule models.

For example :

```
concept( n. of attributes, attributes names,
         dependencies, contributions, meaning).
```

--- Conceptual network, connecting and naming all particular domain concepts.

For example :

```
therapy( 3, attrbtrp(_,_,_), [disease,pacient],
         [none],[medical advice]).
```

--- Concepts tree, particularizing for each concept all its attributes characteristics. Note that the second argument of the predicate representing a specific concept is a new data structure whose instantiations represent all the attributes characteristics under that concept. After having selected a specific concept predicate, its attributes predicates are directly accessed by means of that second attribute.

For example :

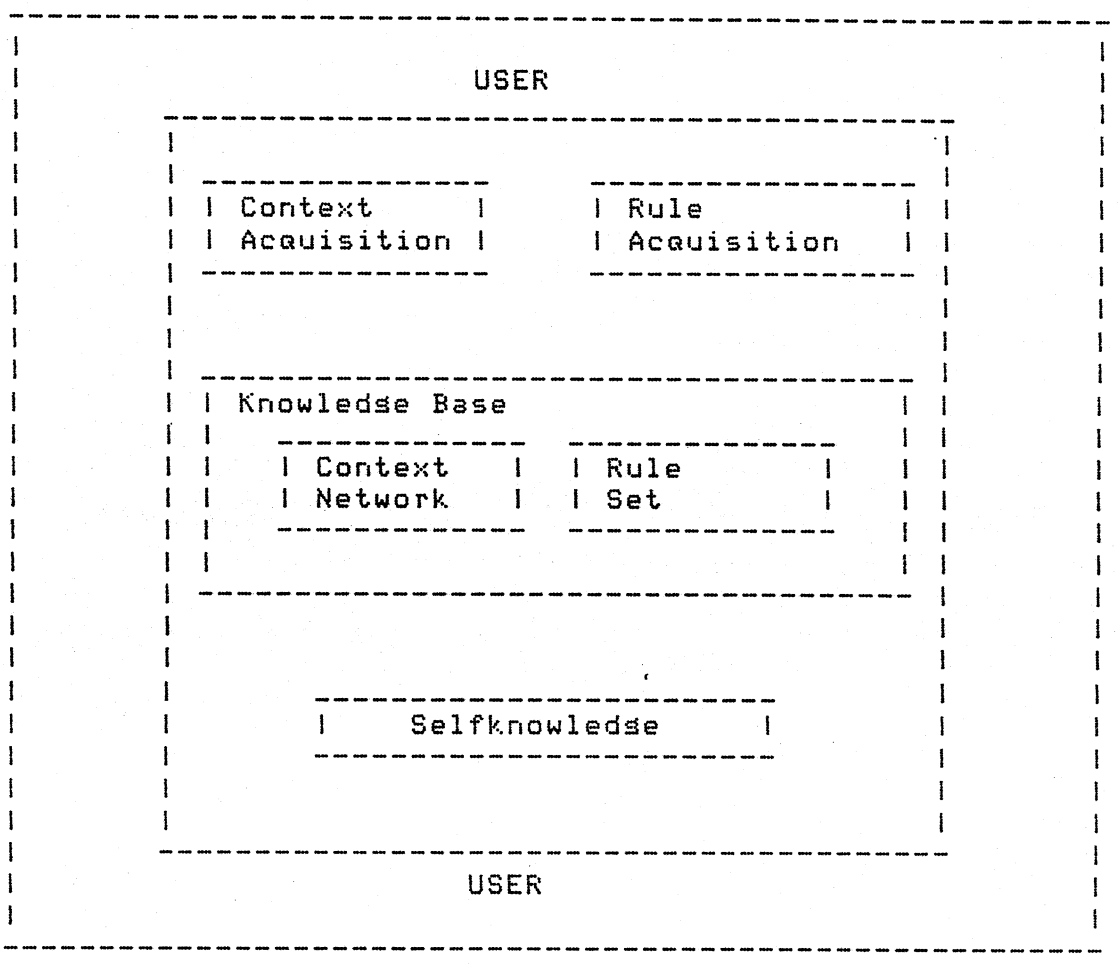
```
attrtrp( 'attribute name', 'attr. unities',
                           'how is obtained').
attrtrp('attribute name',... ).
...
```

All this contextual knowledge is once for all entered by the expert and then it guides the consultation subsystem over the protocol session. It also gives the structure of knowledge which can be consulted by the user.

This feature which is called metaknowledge or selfknowledge represents a kind of introspective capability of knowing about its own knowledge and showing it. Of course that Prolog's

property of programs seen as data facilitates this possibility.

This kind of knowledge archetypes is also important to check rule acquisition. In fact, the other knowledge base major component is a set of Production Rules each of which embodies a chunk of expert domain knowledge, drawing inferences from some concept attribute value to other one. We can display the KBAS module as following :



Rule acquisition is done in a flexible stylized language where several words are recognized as operators (where, if, else, or, and, not...), others as concept attribute's names and its specific values, and finally other ones as functions like (equal, greater,lesser,different...). It is obvious that Prolog non unit clauses (declarativly : 'Head' if 'body') are clearly suitable to represent production rules ('conclusion' if 'premises').

After being checked for syntatic and mostly semantic consistency (regarding contextual network) production rules are precompiled in an internally procedural form and are modularly integrated. A Prolog procedure (a set of Prolog clauses) looks into an intermediate file to where the inputed rule was sent reads it and build the correspondent new clause.

During such an operation optimizations are done to avoid duplication of evaluable predicates into rule's bodies in case of complicated concatenations of boolean operators ("or's" inside "and's", "ored" branches with same concept attribute's name), and so improving rule's fireing efficiency.

Note, however, that they can be displayed again in the same form as they had been entered, by means of a decompilation module which translates them back from internal representation to the more friendly input language.

Rules which shall capture experts knowledge as near as possible its primitive form, must, if necessary, enable several conclusions and complex permises. This is a clear and natural way of trying to prove several conclusions (goals) in a pre-determinate sequence.

When the expert gives such a complex rule he means that those alternative conclusions of the rule are connected and if the first one fails, second shall imediatly be tried and so on. This kind of agregate often corresponds to knowledge organization in expert's head. What I mean is that non--determinism is not always the best way to deal with knowledge representation.

Note that later, if the system keeps track of its successfully fired rules it will know not only which was the right conclusion but also that other ones appearing before in the same rule were tried and failed.

So, rules space is organized as a set of rules subsets (also known as knowledge sources), regarding each concept's attribute, and each rule can either have several alternative conclusions or only one.

This is a very nice improvement. In fact, other well known systems like Emycin [VME], only have very simple rules with one conclusion and a conjunction of single permises.

Our rules can, if necessary, be much more complete as seen in the following example :

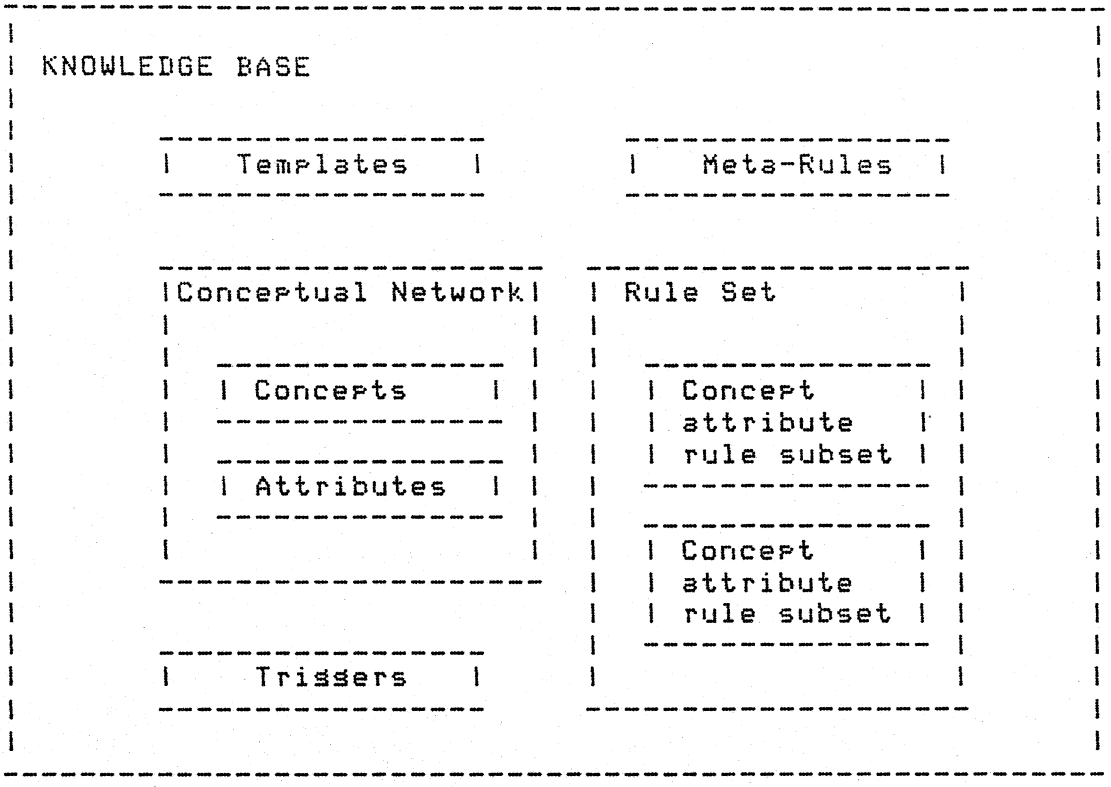
```
disease name = influenza

        if
                syndrome name = headache
                        and
                syndrome duration > two days
                        and ...
                or
                symptom name = feever
                        and
                (symptom intensity > moderate
                                or ...)
                        and ...

else
        disease name = ...

        if ...
```

Rules can also be inspected about theirs components, and concept attributes which contribute for them can also be retrieved, on demand, by searching into theirs bodies.

Several other anotations like rule's name, author and data are also given.

System has a few meta-rules. Meta-rules embodie knowledge about rules themselves. One meta-rule asks the expert if he wants to encode such an ordered alternative conclusions and, if it is the case, instructs him about rule's form.

Knowledge Base is structured as followins :



.Summarizing

When a session besins and if user's (a certain domain expert) password enables him to access knowledge base buildins he can either enter a completly new knowledge base or consult an old one for updatins.

Updates are kept in a separate file to be consulted alternativly or if modifications are definitive the new file will contain the old one already updated.

During knowledge base buildins, the user is on a hierarchical way asked for :

-- concept's names, theirs short mnemonics, theirs mutual relations, number of attributes, meanins.

-- concept attribute's names, possible values, its unities (if mesurable), how shall they be known to the system.

-- Which are the attributes values (if there are any) whose presence is able to directly senerate a set of hypothesis,

expectations to be verified later on. Such an information (called 'triggers') will be responsable for win of efficiency during evaluation process of the consultation session, approaching once more experts way of reasoning.

Simple or complex rules are entered, checked, compiled, retrieved and organized into knowledge sources.

Guided modifications can be done either into the rules or concept network.

If Selfknowledge module is activated all knowledge base information can be accessed (including rules bodies) and clearly presented.

Once again Prolog and its assotiated Horn clause logic it is a very natural formalism to encode knowledge either facts or rules.

.Consultation and Inference mechanisms

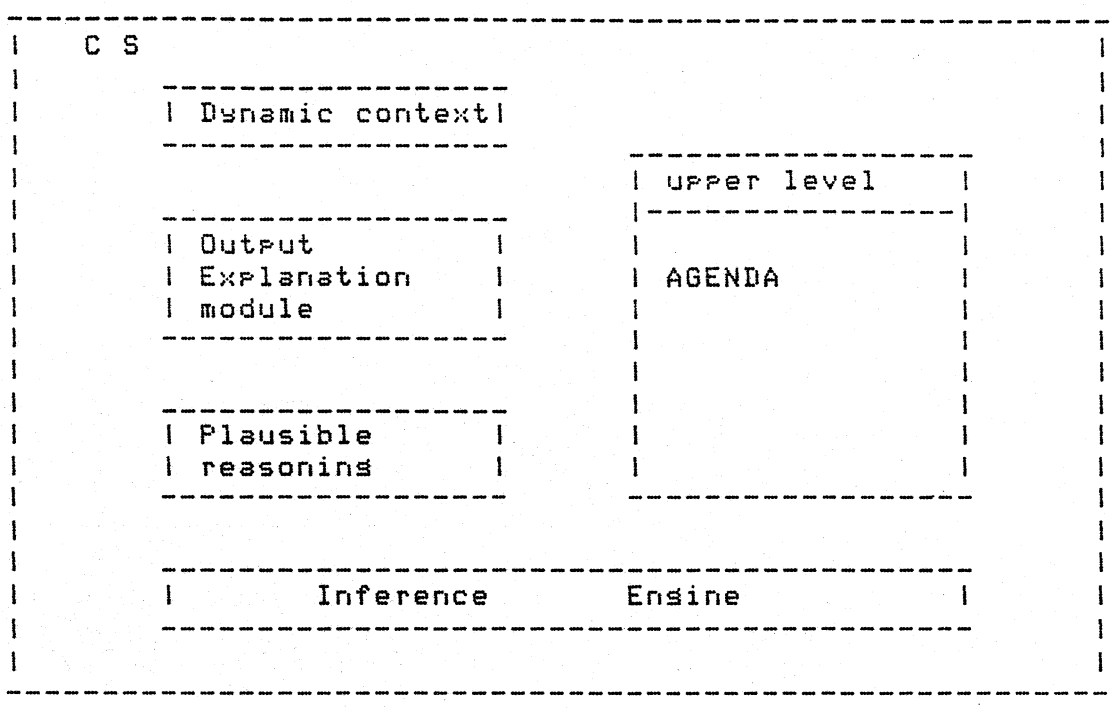
Consultation subsystem, a module under development, uses a selected Knowledge Base (for example a certain medical field), previously build up by means of KBAS module, to interact properly with the user. Note that any user (and not only domain experts) are now able to use Consultation system.

In each session, as result of such an interaction, all needed information is collected. A dynamic context network is built up accordingly to the static one, and the conclusion is reached in an efficient way using production rules selected from respective knowledge sources. Explanations are also obtained.

From KBAS the system already knows possible top goals (ex. therapy, disease). It also knows the set of "triggers" (hypothesis generators), and between them those whose values shall be asked for at the beginins of the session.

So, a protocular session when consultation starts, collects these possible highly discriminatory information.

At that moment inference engine is applied to these two extremes of the space problem (data and top goal) to find the solution.



.Inference methods

Expert systems can be used in a number of applications so diferentiated as diagnosis, plan, design and education among others.

It becomes very difficult to present enough generalized techniques to cover all these possibilities. Nevertheless, several reasoning methods should be available to give versatility to each particular class of systems.

As Prolog is our unique implementation language, the sussested strategy is backward chaining (goal directed) and depth first with backtraking. This is also the strategy used by well known ES like Mycin (Shortliffe) as well as its derived essential system (EMYCIN).

However if search space is very large (namely if the proof tree has a big amount of parallel branches near the root), simple top down becomes inefficient and other search methods must be pursued.

Our inference engine takes advantage of initial data token from the user and of Knowledge Base information about hypothesis generators to prune, earlier, the search space tree. A little cyclic interpreter takes these data, asks KB

for appropriate (matched with that data) 'triggers' and, if they exist, climbs up the tree, guessing some hypothesis and trying to prove them all the possible ways around. If it is the case, these hypothesis (now intermediate conclusions), are asserted in an 'agenda', a globally accessible data structure, and the cycle is repeated with these asserted facts and other possible 'triggers'.

Very many initial possibilities can so be discarded and search space will become more workable with this kind of hierarchical generate and test method.

When this cycle is over the inference engine still keeps the main objective it wants to achieve (top goal) and 'agenda' has all proved intermediate conclusions. At this moment the system can choose between two reasoning methods : forward chaining from asserted data or backward chaining from top goal till it meets assertions in 'agenda ' or in the data base.

'Agenda' has a segmented structure with an individualized upper level. So, as forward chaining proceeds, not every assertion in the 'agenda' is taken into account but only those in upper level, representing nodes nearer top goal, avoiding combinatorial explosion of search paths. In each cycle nodes directly connected with those ones but one step up the tree are tryed to be proved. This implies a reconfiguration of the upper level 'agenda', deleting assertions from which the cycle had started and the assertion of proved new ones. Such a process may continue till top goal is reached if desired.

If backward chaining is choosen, which depends on efficiency considerations, the interpreter looks for the top goal clause, try to prove its body and so on recursively till it meets data or already proved facts on the 'agenda'.

These methods imply, of course, that 'agenda' access mediates each decision step.

.Explanations

Mixing hypothesis generation with forward and backward chaining mechanisms makes explanation task not so easy as if it was only one direction inference (for example top down like in Orbi or Mycin).

During computation all proved steps are carried on a special data structured argument or conveniently asserted in the 'agenda'. They are connected by different constructs depending how they were infered or if they are alternative paths. We keep this executed code, a kind of program's trace, and then we look at it as data to be manipulated. Prolog's

programs declarativity is once more very much appreciated.

An appropriate output procedure deals with these constructs building up an enough understandable output explanation, where we can distinguish between input data, suessed information and step by step infered conclusions.

Already used in Orbi the system will also dispose of another interpreter which discriminates inside rule's body deterministic parts from non-deterministic ones, computing the former and delaying the latter. This interpreter, like other ones into the system, is of course written in Prolog.

An exemple of a compound explanation will be :

Explaned answer for "X" :

"E" is a valid intermediate conclusion because :

"A" was given for you
and "B" is a fact
and to the question "C" you answerd "D"

still another explanation for "E" is :
I already know "F"
and the truth of "F" implies "E"

and finally from "E" I can deduce "X".

.Conclusion

Under development is a kit of programs to build up Expert Systems in several knowledge domains. At present we focused our atention at system architecture and convenient knowledge representation structures, selfknowledge inference mechanisms and explanation capabilities.

Other components like natural language interface and plausible reasoning models will be later on implemented.

We propose to combine several knowledge representation structures as semantic networks and production rules, to use a meta-knowledge module to guide user's consultation, to apply hypothesis generation and bidirectional inference. Composed but understandable explanations are already suggested.

Prolog is our unique implementation language and a very much suitable one.

References

[PER] ORBI- an Expert System for environmental resource evaluation through Natural Language
Pereira,L. ; Oliveira E. ; Sabatier P.
Proceedings First International Logic Programming Conference, Marseille 1982.

[VME] The Emycin manual
Van Melle ; Scott ; Bennet ; Peairs
Department of Computer Science, Stanford University 1981.