

CONTROLLING KNOWLEDGE DEDUCTION
IN A DECLARATIVE APPROACH

Hervé Gallaire
ENSAE - CERT
2, Avenue Edouard Belin
31055 Toulouse Cedex
France

Claudine Lasserre
ENSAE
10, Avenue Edouard Belin
31055 Toulouse Cedex
France

This paper^{*} is concerned with the problem of introducing user-defined heuristic control over the deduction process in a knowledge based system. Control information is seen as being just another piece of data, separated from the knowledge base the use of which it controls and from the theorem prover. A declarative approach to control expression was retained. Control information is embedded in user's metarules taken from the metalanguage presented in this paper. This metalanguage and a theorem prover interpreting it have been implemented. Examples of its use are given and discussed.

1. INTRODUCTION

The process of deducing information from a knowledge base and from deduction rules is central to many fields : proof theory, problem solvers, expert systems in many areas. Whatever the approach followed, be it more declarative or more procedural, the major problem encountered is that of guiding the deduction process, namely the control problem.

Expressing control over the deduction process is but one aspect of metaknowledge, as shown in [3] ; we give here an application-independent formalism for expressing application-dependent control, which is a way of heuristically guiding a general process. The metarules presented here for that purpose have been developed in a declarative context and as such come close to ideas from [2] ; we were influenced by [5] and [8] too.

The paper is organized as follows. Section 2 presents the framework of this study and the PROLOG system we use for knowledge and metaknowledge expression. Section 3 gives an overview of the possible levels and means for metaknowledge expression while section 4 presents the metalanguage we have defined and examples of its use.

2. THE DERIVATION PROCESS

2.1. A point of view for deduction control

Reasoning strategies have been of main concern in many fields of A.I investigated at CERT [9], [12]. For instance Data Base systems need to be enhanced with deduction of implicit information from the explicit one or with checking of integrity constraints to be satisfied by the stored data ; CAD systems need problem solving techniques for exhibiting some kind of intelligence in the design process [10], etc.

Most of the systems built at CERT rely on a declarative approach [15]. But we do not deny the practical interest of more procedural approaches [15]. In these studies the choice was guided by the interest in separating knowledge expression from the expression of algorithmic information on how to use it, and in studying this relationship ; a declarative approach, together with the use of logic as a formalism, proved to provide a very interesting basis for that. Its benefits in knowledge representation - ease of specification, clarity, adaptability, ease of modification of the knowledge base, power of the deductive or proof mechanism - are well known. Besides, we don't believe the proof mechanism to be more complex, although it is more powerful, than dynamic pattern-directed invocation ; it is reasonably efficient when improvements are made in specific areas such as clause indexing and retrieval.

As the declarative approach makes it necessary

^{*} Study sponsored in part by CNRS and DGRST.

to express "how to use what we know" ie meta-knowledge, it gives a good framework for experimenting not only the different possible levels of metaknowledge expression but its different effects on the same data too. Here stands the basic point of this paper, namely that metarules - ie rules on how to use the knowledge base [2] - can actually be expressed in a declarative manner separately from the knowledge the use of which they control. It is important to note that the necessity of expressing control is not an additional work which could be viewed as a default of the declarative approach : as argued in [5], the possibility for the user to mix control to his program in the procedural approach will raise a problem of interpreter control instead of a program control problem.

This study on control of the deduction mechanism rests upon the logical system PROLOG as a formalism and inference mechanism both for rules and metarules. Ideas developed here could as well be carried over to other systems, such as rule-based systems, production systems, provided they offer features similar to the second-order logic ones that PROLOG gives.

2.2. Notations and interpretations

We shall use the First order predicate calculus (FOPC) under clausal form, restricted to Horn clauses [8] and assume the reader to be familiar with theorem proving [1]. We use the following notation :

- (1) $+A -B_1.. -B_n$ stands for $(B_1 \wedge .. \wedge B_n) \rightarrow A$
- (2) $+A$ stands for A
- (3) $-B_1.. -B_n$ stands for $\neg (B_1 \wedge .. \wedge B_n)$

Literals are either positive (eg $+A$) or negative (eg $-B_i$). A is the head of the clause, the rest of it is its body.

These Horn clauses can be interpreted in many different ways. Their standard interpretation is : if $x, y, ..$ are all variables in the clause, then (1) asserts that for all $x, y, ..$ A if B_1 and ... and B_n ; (2) asserts that for all $x, y, ..$ A ; and (3) asserts that for no $x, y, ..$ B_1 and ... B_n (i.e. $\neg \exists x \exists y .. B_1 \wedge .. \wedge B_n$). Other interpretations are possible, in terms of problem solving ((1) being an operator, (2) an assertion, (3) a problem), or of programming languages. ((1) and (2) being procedures, (3) a procedure call). We shall stick to the logic language as much as possible in the sequel.

As for the proof procedure we shall only consider derivation or refutation procedures, based on resolution, restricted to linear strategies

starting from purely negative clauses. At each step i of the derivation we are given the first (negative) parent clause C_i (FPCL) ; we have to choose the literal $-C_{ij}$ in C_i as the literal to be resolved upon at this step (C_{ij} is called SEL) ; we also have to choose among all clauses those whose head unifies with C_{ij} (making the UNIFCLSET) ; finally we have to choose in UNIFCLSET the second parent clause (SELCL) ; resolution yields resolvent C_i' ; C_{ij} is said to be the parent of the literals in the body of SELCL. All the above choices will be guided by a strategy.

2.3. Background on PROLOG

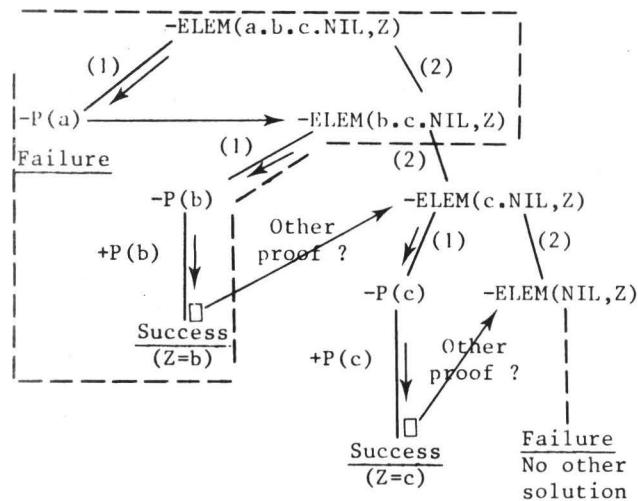
PROLOG [13] is a theorem prover which takes as input a set of Horn clauses of type (1) and (2) above and a theorem to be proved, or actually its negation of type (3). It uses a static strategy complete for Horn clauses, namely LUSH resolution [6] ; it is "top-down", starting from the purely negative clause it is given ; it chooses as SEL the leftmost literal in FPCL, working "depth-first", and chooses clauses from UNIFCLSET in the exact order they were input to the theorem prover by the user. Further, the process is controlled by an automatic backtracking mechanism activated in case of incorrect clause choice and in order to find all proofs of the alleged theorem. Very important to our work is that, due to LUSH resolution results, any other SEL selection strategy would preserve completeness.

We must note that PROLOG implementation is not complete, due to the unbound depth-first mechanism and to the lack of loop checks. But PROLOG exhibits great advantages for metaknowledge expression : it allows using clauses and predicates as terms, i.e as arguments of other predicates ; through built-in predicates it offers access to input-output, to the proof tree (ETAT predicate) and to the various parts of clauses and terms (VAR, UNIV predicates) ; one can also alter the set of axioms (ADD and SUPP predicates) ; it is also possible to alter the standard backtrack step by eliminating return points : any selection of the '/' predicate in a SELCL will suppress all remaining choices for all the already solved literals in SELCL and for their parent.

Example : the following is a set of axioms for proving that Z is an element of list L satisfying predicate P . The '.' is the concatenation symbol and NIL is the empty list.

- (1) $+ELEM(X, Y, X) \quad -P(X)$
- (2) $+ELEM(X, Y, Z) \quad -ELEM(Y, Z)$

When used to prove $\exists Z \text{ ELEM}(L, Z)$ axiom (1) will unify Z and the first element of the given list L ; then if Z satisfies P , the theorem will have been proved (the value of Z being the answer of PROLOG). Due to PROLOG strategy for choosing SELCL, axiom (2) is used in case axiom (1) does not yield the empty clause, i.e. when $P(X)$ cannot be proved; in that case a new theorem is to be proved, namely $\exists Z \text{ ELEM}(Y, Z)$, where Y is the rest of the list. If $+P(b)$, $+P(c)$ are axioms and the theorem to prove is $\exists Z \text{ ELEM}(a.b.c.NIL, Z)$, the derivation tree will be:



Recall that PROLOG gives all possible proofs; hence when success is recorded an alternative is looked for. In this tree the arrows indicate the order in which the nodes of the tree are built. If one wanted only one proof of that theorem, it would be simple to modify (1) and to write it $+ELEM(X.Y, X) - P(X) -/$. In that case we would get a tree limited to the one enclosed in the box above.

3. PARAMETERS OF THE DERIVATION PROCESS

A choice strategy rests upon a selection function. Such a function may or may not depend on the object on which it bears; for instance it may depend on the specific clause from which to extract SEL, or it may not depend on the specific UNIFCLSET from which to extract SELCL. Similarly it may or may not depend on the context of the object on which it bears; by context we mean the derivation tree or part of it.

The information needed by the selection function may be either implicit or explicitly specified by the user. Autonomous selection functions use implicit information (which may even be void, eg

the leftmost-literal-first rule seen in PROLOG) as in the literal-with-the-least-number-of-instantiated-variables-first rule, or in the shortest-clause-first rule, etc. Such functions will not be discussed in this paper, although an interactive choice of such strategies is offered to the user in some systems.

This paper is precisely concerned with the expression of information to be used by selection functions; we shall only deal with the means of expression, not with designing specific criteria. Important to this work is the notion of context; to express context information, the user must choose between a local context (part of the proof tree, eg the parent of a literal, its parameters, its descendants) and a more global one (ancestors of a literal, the whole proof tree itself). Although the declarative approach that we follow, especially the logic one, seems to be well suited to that latter choice through the notion of proof, we have not investigated it, thus restricting ourselves to local context expression. Neither shall we discuss the pragmatic control expression that clause writing always conveys [7].

There are two basic ways for expressing control of the deduction process.

a. As is typical of the procedural approach, control information can be mixed with the knowledge base, ie with the application data (clauses) used in the derivation process. For instance the user can modify his set of clauses so that the order of his original clauses in the UNIFCLSET they belong to becomes dependent on SEL. We do not reject such an approach; its systematic use is reported in [9].

b. More coherently with the declarative approach, one can separate control information from the knowledge base; at the user level this becomes another piece of data, ie a set of clauses expressing a heuristic control. This is the choice that was made here.

We shall now present our metalanguage for stating the metarules expressing this heuristic control. It uses content-directed invocation [2] heavily - and we feel we have the correct tools for it. Our metalanguage is coherent with the base language [5].

4. A PROPOSAL FOR A METALANGUAGE

There are two types of metarules in this language, expressing control either over the choice of SEL or over the choice of SELCL; for the user

writing them, such metarules are true assertions, in the logical sense, about priorities and orderings as explained below ; they will be used as axioms by the theorem prover.

4.1. Ordering of UNIFCLSET

The METAL metarules to be described here have as primary purpose the expression of assertions on priorities of individual clauses in UNIFCLSET, SELCL being the clause of higher priority. Then let P, R be terms or variables, let L be a term $L_1, L_2 \dots L_r, \text{NIL}$ or a variable (see 2.3 for the ',' notation and for NIL) ; let $C_1, \dots, C_n$ be positive literals ; then Metarule

+ METAL(P, L, R, K) -C₁ ... -C_n

asserts that any user's clause matching (unifying) with clause +P-L₁-...-L_r has priority K in the UNIFCLSET it belongs to, provided R matches the parent of SEL and the existential closure of $C_1 \wedge \dots \wedge C_n$ can be proved ; in the sequel we talk more simply of the proof of $C_1 \wedge \dots \wedge C_n$. We must remark that :

- L_i, R, P are terms in METAL but literals in user's clauses (second order feature).

- when any of P, L is a variable, the process of matching will instantiate it ; similarly when R is a variable it is instantiated to the value of SEL ; this feature allows to have metarules applying to many user's clauses, the target clauses, thus avoiding much redundancy, and making it easier to write metarules by specifying just what is necessary to match the target clause(s).

- the matching process is constrained not to instantiate any uninstantiated variable of the target clauses or of the parent of SEL at the time matching takes place ; this restriction, enforced by the theorem prover means that a metarule shouldn't be used when it aims at a special case of a target clause. This restriction implies that during the proof of $C_1 \wedge \dots \wedge C_n$ no variable in P, L, R is instantiated.

- there is a default metarule asserting a MEAN priority value for clauses which aren't target of any metarule : +METAL(P, L, R, MEAN) where P, L, R are variables.

- the set of metarules may be conflicting ie a clause might be the target of two metarules ; as no meta-metarule has been introduced yet, we cannot handle it differently from PROLOG, ie the first metarule that is proved is retained.

- the C_i predicates can be defined by any valid

PROLOG set of axioms.

Example : +METAL(P(X,a), L, R(X,Y), 0)
-PRED1(X,a) -PRED2(Y)
-PRED3(L,F)

This metarule asserts that priority 0 is the priority of any rule the head of which unifies with P(X,a), the parent of SEL unifying with R(X,Y), provided the conditions PRED are provable. The matching process must comply with the above restrictions - eg as 'a' is constant, X variable, this is not allowed to match P(T,U) where T,U are variables, but P(b,a) would pass the test ; during a proof of PRED1(X,a) it is not allowed to instantiate X more than it is already.

4.2. Choice of SEL

We are to introduce local context evaluation for SEL selection ; there are two selection factors :

- backtracking blocking corresponding to a dynamic '/' introduction

- literal freezing preventing the theorem prover from choosing as SEL a frozen literal ; this feature, the most interesting of all features we present, allows for very dynamic control regimes.

Let P, L, C₁, ..., C_n be as described for METAL metarules ; let I and F be terms to be described below ; then

+METAX(P, L, I, F) -C₁ ... -C_n

asserts that if +P-L₁-...-L_r matches the candidate SELCL, and if $C_1 \wedge \dots \wedge C_n$ is provable, then :

- if I is the term INH(I₁...I_m, NIL) where each I_j matches some L_h, any backtracking on L_h and on its parent will be blocked ; INH (NIL) blocks P only ; if I is NIL no blocking occurs.

- if F is the term FRZ(F₁, V₁)....FRZ(F_s, V_s).NIL where each F_j matches some L_h, then L_h will be frozen as long as none of its arguments appearing in list V_j is instantiated (ie anything but a pure variable) ; if F is NIL no literal is to be frozen.

We must remark that much of what was noted for METAL metarules applies to METAX ones and is not repeated here, especially for instantiation. Also, there might be cases where all literals are frozen ; we unblock the situation by choosing the leftmost one (default rule). We are now to see examples and to examine more deeply the freezing capability, already investigated but solved differently by Colmerauer [16].

Examples on freezing literals

The following set of PROLOG clauses can be used to prove that two trees X,Y have identical lists of leaves when traversed from left to right :

```
+IDLF(X,Y)-LF(X,S)-LF(Y,S)
+LF(X,X.NIL) - ATOM(X)
+LF(U.V,Z) -LF(U,X) -LF(V,Y) - CONC(X,Y,Z)
+CONC(NIL,Y,Y)
+CONC(A.Z,Y,A.V) -CONC(Z,Y,V)
```

If it has been given clauses ATOM(a), ATOM(b), ATOM(c) and if IDLF((a.b).c, a.(b.c)) is a theorem to be proved, PROLOG will succeed ; however the control regime of this proof is quite unsatisfactory for, if it were to fail, much useless work might have been carried out before checking anything. This is a classical example for coroutine need, as a coroutine would allow checking each leaf as soon as it is uncovered in Xor Y. The following metarules, together with the default rule, will do :

```
+METAX(P,L,NIL,FRZ(LF(X,S),S.NIL).NIL)
+METAX(P,L,NIL,FRZ(CONC(U,V,Z),U.Z.NIL).NIL)
```

The first metarule will prevent the theorem prover from choosing LF(X,S) as SEL as long as S is not instantiated ; the second one will do similarly for CONC(U,V,Z) whenever both U and Z are uninstantiated.

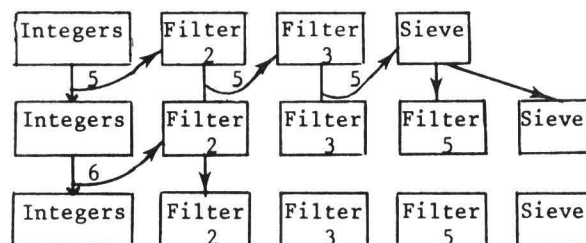
A different and even more interesting example is the classical Sieve of Eratosthenes problem. The following is the list of PROLOG clauses that might be necessary to prove Primes(X) where X is the list of prime numbers

```
+Primes(2.X) -Integers(2.Y) -Sieve(2.Y,X)
+Integers(P.(Q.R)) -Succ(P,Q) -Integers(Q.R)
+Sieve(P.IN,P.PR)-Filter(P,IN,NEW)-Sieve(NEW,PR)
+Filter(P,Q.IN,OUT)-Mult(P,Q)-Filter(P,IN,OUT)
+Filter(P,Q.IN,Q.OUT)-Filter(P,IN,OUT)
```

The Succ(P,Q) predicate is satisfied iff the value of Q is the value of P plus 1 ; the Mult(P,Q) predicate is satisfied iff Q is a multiple of P (definitions not given here for lack of space). As argued in [11] where it is presented, due to PROLOG selection rule this neat example will not work as it is. However it is quite easy to fix with the following :

```
+METAX(X,L,NIL,FRZ(Integers(S.T),T.NIL).NIL)
+METAX(X,L,NIL,FRZ(Sieve(U,V),U.NIL).NIL)
+METAX(X,L,NIL,FRZ(Filter(M,N,K),N.NIL).NIL)
```

Snapshots will explain how the proof proceeds :



4.3 - Implementation principles

There is no major problem in such an implementation, although particular attention is to be paid to allow correct backtracking, fair selection of frozen literals. The theorem prover has been coded in PROLOG mainly for ease of modification and for its unification algorithm. As we piled up levels of interpretation the result is rather inefficient, but our goal is attained. We emphasize how useful both the non first-order features of PROLOG and the possibility to examine its own data structures, i.e. terms and clauses, are. Due to the very simple access to those structures (UNIV, VAR, ...) it is possible to build powerful mechanisms.

Of course this theorem prover is not a major innovation in the field ; one can argue that it behaves with respect to PROLOG in a manner quite similar to CONNIVER's behaviour with respect to MICROPLANNER. This is basically true from the viewpoint of the theorem prover ; what we gave back to the user is some control over lists of possibilities, over backtrack, over the context mechanism [14]. This similarity is only superficial. We give more to the user because what he can examine in our case is not just a list of possibilities, but it is a list of methods in the CONNIVER sense ; the method body can be scrutinized, modified, reordered whereas it cannot happen with CONNIVER ; this is due to the content-directed invocation feature that we mentioned earlier and of which we saw some important effects in 4.2. We recognize that our context handling mechanisms, which are roughly the SUPP and ADD predicates of PROLOG, are not as powerful as CONNIVER's. Our goal was different and this remains a possible extension for this work.

5. DISCUSSION

The points we want to stress are the following :

a. User responsibility - Metarules give the user much freedom ; a correct use of metarules goes through an understanding of how the derivation process should go ; then the user will be tempted to use this knowledge and modify his own

algorithm so as to take advantage of this understanding directly. This is the reason why we did not give real examples for METAL metarules nor for backtracking use of METAX metarules ; simple examples can easily be fixed without metarules, although this usually introduces much redundancy and awkwardness in the set of axioms. From our studies in plan generation, we are aware of cases where a '/' could be dynamically introduced with much benefit.

b. Extensions - Any time a language is defined, one may propose extensions to it. Among all the extensions we have been thinking of, one is to allow user-defined FRZ predicates ; such a need has been found in another plan generation program where redundant tests could be avoided by properly defined freezing conditions ; such an extension is quite simple to incorporate to our theorem prover.

c. Related topics - Much of the interest of metarules comes from the dynamic ordering of literals (4.2) ; clearly this is related to the many producer-consumer discussions, to the call by opportunity or call by necessity choice ; the Primes problem is discussed in [11] and in a different framework in [4] ; we believe the simple tools given here compare favorably with those given there. We use a data flow mechanism (instantiation) ; more explicit control mechanisms such as message passing have not been looked for. Note that it is possible to define and implement a theorem prover which could take advantage of the connections between variables to select literals ; for instance, in the Prime example the instantiation of a variable unfreezes just one literal, and usually a theorem prover can watch such instantiations. Expressing specific properties of predicates, e.g. that they are true functions, is not covered here although it is an important domain to be investigated. We believe "intelligent" backtracking to be an important topic deserving more attention ; we only note that to be intelligent it must be prepared during the derivation process ; the tools we give are but one step in that direction. Finally interactive control expression has not been covered at all.

ACKNOWLEDGMENTS

We would like to thank A. FAHMY for the many discussions we had with him on this topic and on PROLOG use. P. HAYES comments on this paper are gratefully acknowledged.

REFERENCES

[1] Chang, C.L. and Lee, R. "Symbolic Logic and Mechanical Theorem Proving" Academic Press, 1973

[2] Davis, R. "Generalized Procedure Calling and Content Directed Invocation." In Proc. Symposium on AI and Programming Languages, Rochester, August 1977, p 45-54

[3] Davis, R. and Buchanan, B.G. "Metalevel knowledge Overview and Applications." In Proc. IJCAI-77, MIT, Mass., August 1977, pp 920-927.

[4] Friedman, D. and Wise, D. "Unbounded Computational Structures." Software, Practice and Experience. Vol. 8, pp 407-416 (1978).

[5] Hayes, P. "In Defence of Logic." In Proc. IJCAI-77, pp 559-565.

[6] Hill, R. "Lush Resolution and its Completeness." DCL-Memo 78, University of Edinburgh, Aug. 1974.

[7] Kowalski, R. "Algorithm = Logic + Control." Depart. of Computing and Control. Imperial College, London, 1976.

[8] Kowalski, R. "Logic for Problem Solving" DCL-memo 75, University of Edinburgh, 1974.

[9] Lasserre, C. "Apport de la logique mathématique dans les systèmes de décision en robotique." Thèse Université Paul Sabatier, Toulouse, 1978.

[10] Latombe, J.C. (Ed.) "Artificial Intelligence and Pattern Recognition in Computer Aided Design." North Holland, 1978.

[11] Mc Cabe, F.G. "Programmer's Guide to IC-PROLOG 0.7." Department of Computing and Control. Imperial College, London, 1979.

[12] Nicolas, J.M. and Gallaire, H. "Data Base-Theory vs Interpretation." In Logic and Data Bases, Plenum Pub. Co, 1978.

[13] Roussel, Ph. "PROLOG : Manuel de Référence et d'utilisation." Groupe d'Intelligence Artificielle. UER de Marseille-Luminy.

[14] Sussman, G. and Mc Dermott, D. "From Planner to Conniver." In Proc. FJCC 1972, pp 1171-1179.

[15] Winograd, T. "Frame Representations and the declarative procedural controversy" In Representation and Understanding. Academic Press, 1975.

[16] Colmerauer, A. Personal Communication.