GALLAIRE82
Gallaire, H. and Lasserre, C.; "Metalevel Control for Logic Programs", in "Logic Programming" by Tarnlund and Clarke, pp. 173-185, London, Academic Press, 1982 Metaknowledge is knowledge the programmer has about a problem which is beyond the straight facts and logical relationships of the problem. This paper deals with methods of incorporating that metaknowledge into PROLOG-like Horn clause languages to improve their searching and goal resolution efficiency. Specifically, the syntax for a separate level of "Metarules" is introduced which controls the goal resolution method of the interpreter. As PROLOG programs encompass larger and more complex problems the incorporation, somehow, of control information becomes a pressing problem which any serious PROLOG programmer must address. Various methods of incorporating "Metaknowledge" into PROLOG programs are: Pragmatic Control: This consists of physically re-ordering the clauses in a program to help the fixed strategy of the interpreter. For example, the most easily satisfied subgoals in a predicate can be leftmost. Explicit Control Incorporated into the Program: The most common form of this are the cut "!" and fail predicates which modify the backtracking control stack. Recently control constructs which delay the attempt to satisfy a subgoal until one or more of its variables is bound have been introduced. This tends to modify the strict left to right satisfaction of subgoals and improves efficiency by reducing the number of attempted solutions. Explicit control Separated from the Program: This approach uses metaknowledge directly in the form of "Metarules" which are syntactically separate from the PROLOG program. The interpreter uses both these levels of knowledge and runs special search interpretation routines according to the actions specified by the Metarules. Metarules have the advantage of clarifying PROLOG programs by fully separating logic and control and supply a more elaborate set of control structures. The general form of a metarule is

```
+Action-Condition1-...-ConditionN
```

The predicate "Action" is a system metapredicate expressing an action on the derivation process. "Condition" is a system or user defined predicate in a program. A metarule describes an action to be undertaken by the interpreter whenever the interpreter focuses its attention on an object involved in the metarule. The syntax presented in the paper is flexible enough to designate virtually any predicate and allows position or content directed invocation. Other Metarules include those which limit levels of recursion, recover workspace, turn of metarule control, etc... Consider the following Grandfather example:

```
+grandfather(x,y)-father(x,z)-father(z,y) +father(paul,pierre)
```

If the problem is to solve "-?grandfather(x,paul)" it is more efficient to select the second subgoal "-father(z,paul)' to start the satisfaction process. Consider the Metarule:

```
+NEED(father(u,v))-OR(INST(u),INST(v))
```

where: OR is defined as logical OR, INST is true when its parameter is bound and the NEED metapredicate forces clause to have a resource available before it can be selected. In this case searching would begin with the first "father" subgoal with a bound variable. A number of more complex examples and numerous other metapredicates are also presented in the paper. This paper gives a good overview, and references to, much of the work currently being done in trying to incorporate "metaknowledge" into PROLOG like languages. Unfortunately it appears that the metarule concept discussed here has not seen a practical implementation though it seems probable that this is only a matter of time.