MCDERMOTT80

McDermott, D.;
The PROLOG Phenomenon;
ACM SIGART, No. 72, (July 1980), pp 16-20.

Since the late 1960's there has been a strong interest in the computing community for the research and development of an automated, theorm- proving oriented, programming language. Acknowledging the early "AI languages", PLANNER and QA4 proved to be inefficient and hard to control and thus the continuing interest in logic programming lead to the introduction of PROLOG (PROgramming LOGic). McDermott notes the primary characteristics of this language that make it distinct from other conventional programming languages. The strong and weak points of PROLOG are established and clarified through numerous illustrative examples. The article proves to be a fine introduction to PROLOG for those familiar with basic programming language concepts.

The introduction of PLANNER by Carl Hewitt in 1968 featured numerous automated theorm proving concepts: procedural interpretation of deduction, pattern-directed procedure calling, an indexed database of assertions and programs, and non-determinism (backtracking). However, by 1972 PLANNER came to be viewed by the user community as being inefficient and hard to control. In the United States the language was no longer in use; rather, LISP was the language for AI work. Concurrently, ub the early 1970's , Colmerau and Kowlaski rediscovered the procedural interpretation of deduction. This notion was embodied into PROLOG. Based upon the author's research on PROLOG, he concludes "that PROLOG is an interesting and powerful language that deserves to have, and undoubtedly will have, more use in the United States", whose user community has been devoted to the use of LISP in AI oriented areas.

Without spending time on the syntax of PROLOG, the author presents an example program, a quadratic equation solver, using the University of Edinburgh's version of PROLOG. McDermott, through references to this example program, points out the facts that (1) PROLOG uses relations (sets of Horn clauses) instead of LISP-like functions and (2) does not become deeply nested in a sequence of relational calls. He further notes PROLOG's distinction as a programming language in that it does not have an assignment statement; rather, it is through instantiation that the PROLOG interpreter attempts to prove a supplied query as being true.

Through a series of additional examples, the primary concepts and mechanisms of PROLOG are introduced: matching, instantiation, unification, backtracking, and cutting. Matching is used to find the first Horn clause that matches the head of the calling clause. Upon finding such a match, the body of calling non-primitive relation is executed with the variables bound through the pattern match. The failure of a match or execution of the clausal body results in order to carry out another match and instantiation. Notably, recurrent failures cause further backtracking, i.e. a depth first search retrace, which can lead to inefficiency in the use of runtime space and program execution time. The author fails to adequately explain output, especially the meaning of "yes" and "no", in his interesting examples. Yet, he satisfactorily explains an alternative to global backtracking, i.e. tracking and the "wierd loops" that can be generated "through large sections of a program that do not converge to any solution, or do so very slowly."

Having presented the essential tools of PROLOG, the author concludes the article with an explanation of the advantages and disadvantages of the language from his viewpoint. An acknowledgement of the powerful pattern matching mechanism, the diverse list and record structures that can be created, and the availability of "extremely efficient" implementations of PROLOG are among the positive attributes. However, it is emphasized that PROLOG is not programming in logic, except for the simple clauses that can be thought of as first-order implications". Moreover, the author emphatically states that PROLOG can be viewed as an effort to simplify a theorem prover to the point of efficiency. McDermott writes, "I think its inventors may have gone a little too far. They concentrated on implementing one basic idea with more and more efficiency, and have turned their backs on other ideas that may in the long run pay off. In other words, PROLOG may become the FORTRAN of logic-based programming languages". Yet, despite such a strong statement, the author feels that PROLOG "competes very well" with LISP but not NOAH as far as a programming language, "not a problem solver or theorem prover."

While being an interesting approach to the introduction of PROLOG, the author fails to develop an adequate simplicity in some of his examples and thus presents a challenge to the PROLOG novice. In places adequate explanation is given to concepts and personal viewpoints but in others there is an obvious absence of such important clarafication. The article is recommended to those who do not demand a presentation on syntax and elementary programs as parts of an introduction to a specific programming language.