WARREN77
Warren, David H.D., and Pereira, Luis M.; PROLOG: The language and its Implementation Compared with Lisp; Proceedings of the Symposium on Artificial Intelligence and Programming Languages. SIGPLAN Notices 12, 8 (August 1977) and SIGART Newsletter 64 (August 1977) pp. 109-115.

This article discusses a compiler written for the logic programming language PROLOG. The authors try to demonstrate that their compiler for PROLOG generates code that is at least as efficient as compiled LISP code. Since efficiency is one of the major concerns with PROLOG, this suggests that PROLOG may be an appropriate language for Artificial Intelligence work.

The first part of this article describes PROLOG for readers who are not familiar with it. The description is not as good as [MacLennan] or [Kowalski], but it does introduce many terms used in PROLOG that the other two omit. The authors claim that in PROLOG, "clear, readable, concise programs can be written quickly with few errors." PROLOG is much better in this respect than LISP, since LISP functions tend to be longer, to have much more nesting (parentheses), and to use many un-pure forms, such as "prog" and "rplacd". Another advantage of PROLOG is that there is a natural declarative semantics (English-like way to read the statements) in addition to the normal procedural semantics. This makes the programs virtually self documenting. To design a PROLOG program, one typically designs a procedure to check for some condition. This procedure will then also solve for that condition.

The compiler produced by the authors was written entirely in PROLOG. The original version was bootstrapped from an earlier PROLOG interpreter. The compiled code was for the DEC-10 which is well suited for PROLOG due to its inexpensive indirection instructions. The compiler gets large efficiency gains by allocating space for most variables on two stacks, rather than on a heap, so space is easily reclaimed when backtracking. It turns out that the full recursive backtracking nature of PROLOG is rarely used, so most statements can be very inexpensive. Special (non-PROLOG) control statements are provided that allow the programmer to tell the system when clauses are used in this restricted way. Another clever idea is indexing the principal argument of every clause so that the it can be found by a computed goto or hashing rather than searching. The result is a system that executes operations at 1/2 to 2.6 times that of compiled LISP code for the DEC-10, and only takes 5K words for data for the backtracking trail and the 2 stacks.

This article does not address the problems of how to tell the system which clauses to use first, or whether top-down or bottom-up control will be more efficient in the current program. Other articles have suggested that this is a hard problem for PROLOG implementations. In addition, it was not made clear how appropriate PROLOG was for writing large programs, or even how large the compiler was. Much other information that would be very interesting also has been omitted, but the data provided was very understandable and well written. Undoubtably, this article helped convince the Japanese that PROLOG was a viable alternative to LISP for AI programming in the future.