corresponding to *terms* is scanned to ensure that *plus or minus . term . terms* is an alternative when compiling the instruction

→ 1 **unless** *terms* **resolves into** *plus or minus . term* **c** *. terms*

At run time, the tree whose address is in *terms* is inspected to see whether it actually conforms to the right-hand-side. If it does the addresses of its principal sub-trees are planted in *plus or minus*, *term* and *terms*; otherwise control passes to the instruction labelled 1.

Conclusion

The system just described uses **integers** to refer to trees and sub-trees. A theoretically more satisfying scheme using more formal concepts of type and reference for this purpose has been described by Brooker and Rohl (1965). The present scheme, however, has the advantage that it can be fairly easily grafted on to any compiler in which **integers** can serve as store addresses.

The facilities have been part of the Atlas Autocode (AA) compiler since April 1966, and have found a wide variety of uses. A compiler for a language called BB, which is a subset of AA, has been written and debugged, and a compiler for a logical design language is at present under development.

They have found further use in more general data processing jobs. The logging program mentioned earlier has been re-written, and a program to process all the applications for places in this department's honours course has been constructed. This program accepts as data (i) the original application of the student to the U.C.C.A., (ii) any updating material such as offers from other universities and interview assessments, and (iii) interrogatory statements about the state of applications, such as

LIST ALL NAMES INTERVIEWED IN ALPHABETICAL ORDER

Two translation programs have also been written. One converts an AA program from its normal mode into upper case delimiter mode (see Brooker, *et al.*, 1967) and at the same time allows the user to change whatever identifiers he chooses throughout the program. The second converts ALGOL procedures into AA routines. This is in an undeveloped state yet but it is hoped shortly to be able to translate completely some 90% of the published algorithms.

References

BROOKER, R. A., MORRIS, D., and ROHL, J. S. (1967). "Experience with the Compiler Compiler", *The Computer Journal*, Vol. 9, p. 345.

BROOKER, R. A., and ROHL, J. S. (1965). "Simply Partitioned Data Structures", *Proceedings of a Machine Intelligence Workshop 1965* (Ed. Michie, to be published by Oliver and Boyd).

BROOKER, R. A., ROHL, J. S., and CLARK, S. R. (1966). "The Main Features of Atlas Autocode", *The Computer Journal*, Vol. 8, p. 303.

Book Review

A Syntax-Oriented Translator, by P. Z. Ingerman, 1966; 131 pages. (New York: *Academic Press*, 48s.)

It is difficult to review a book the chief deficiencies of which are announced as such by the author by the end of the third paragraph of his preface. Indeed, the author further disarms us by overstating the case against himself and his book; he has not committed hubris and the wrath of the gods will be withheld so that he may live to assay Olympus yet again.

The title well describes the book, which was originally intended "for the home compiler-writer". When the author concluded that people do not write compilers at home he forgot that they may be required to do so in a provincial computer installation, very often a more difficult task. It is thus sad that the book needs a "patient reader". Would it have been possible to produce a simple cookbook, three times the size and five times the value to the trade? In the present overburdened state of education everywhere we need more instruction manuals.

A reader with enough persistence can certainly use this book to help implement a syntax-oriented translator, and to devise the language and machine descriptions for some cases of interest. The major, all too common, defect of the book is the use of the assignment statement of ALGOL and FORTRAN as the worked example. We hope for an author who will choose the COBOL option "Move Corresponding", preferably from a group of computational fields to a display record; or perhaps the "Preserve", "Restore", or "Task" options of PL/1. These are examples where this type of translator might show advantages over the "operator/operand stacks" method of Dijkstra, and Randell and Russell, which is perfectly adequate for arithmetic expressions and perfectly general if designed properly to be table-driven.

More precisely, given a machine with basic software and an arbitrary language not foreseen by the designers of the software to implement on that machine, then the design of the structure of the object program to model the semantics of the language is far more difficult than the syntax analysis of the language. But a generalized syntax analyzer is a worthwhile convenience.

H. D. BAECKER