actual parameter is to be substituted.

42. The structure/number of dimensions or parameters of a formal parameter used in an actual parameter expression does not match its use.
43. A formal parameter used in an actual parameter expression expects an identifier to be substituted for it, but finds instead an expression.

4. Conclusion

This study demonstrates how the static semantic rules of two common programming languages can be specified formally. The static semantic rules of ALGOL60 are much more difficult to specify than those of BASIC, partly because of the ability to mix types within an expression under certain circumstances but not under others (e.g. ÷), and partly because the type and structure of a formal parameter need not be declared.

For the complete specification of ALGOL60, situations were encountered for which the correctness could not be decided from the original report. The implementation of such situations is left to the compiler writer's discretion. However, if one is to standardise programming languages so that a program written in a given high level language has the same syntax and produces the same effect (wherever possible) when run on different machines, one needs to specify all aspects of the language completely. Thus in the case of undecideable situations in ALGOL60 a decision was taken one way or another. Thus the specification of ALGOL60 given here will produce a similar effect to that produced by some ALGOL60 compilers but not by others. It is not suggested that the specification given here should necessarily be the generally accepted one; however, the intention is to show that the complex static semantic rules of ALGOL60 can be expressed in a formal notation which is meaningful to compiler writers and hence assist in the standardisation of such a language.

Besides its effect on standardisation such a formal specification may also be of assistance to the compiler writer from the point of view of the correctness of the compiler he produces by giving him a model of how these aspects of the language can be implemented at the very simplest level. Obviously the compiler writer will have his own ideas as to how best to implement certain features to provide the most efficient system. However, at a time when the correctness of the product is as important as its efficiency, by providing the compiler writer with a simply implementable specification of these aspects of the language, he has a yardstick by which to measure the correctness of his own implementation.

References

BULL, G. M., FREEMAN, W. and GARLAND, S. J. (1973). *Specification for Standard BASIC*, NCC Publications: Manchester.

LEDGARD, H. F. (1969). A Formal System for Defining the Syntax and Semantics of Computer Languages, Ph.D. Thesis, MIT.

LEE, J. A. N. (1972). The Formal Definition of the BASIC Language, *The Computer Journal*, vol. 15, pp. 37-41.

NAUR, P. *et al.* (1963). Revised report on the algorithmic language ALGOL60, *The Computer Journal*, vol. 5, pp. 349-367.

WILLIAMS, M. H. (1978). A Formal Notation for specifying Static Semantic Rules, submitted to *Computer Languages*.

Book reviews

The Complexity of Computational Problem Solving, edited by R. S. Anderssen and R. P. Brent, 1976; 262 pages. (*University of Queensland Press*, £4·70)

The title is mildly ambiguous, since this book is not so much concerned with problem solving in the usual sense (although this is certainly involved) as complexity in all its computational aspects, and understood indeed in a rather broad way to include efficiency in the formulation, programming and debugging phases of program construction. The problems are therefore primarily with respect to the methods of computing and less with the 'external' problem for which the computer is to be used.

An example of an important external problem is the last of the fifteen articles which is called 'the complexity of real-world scheduling problems'. These problems include resource allocation and assignment, timetabling, job/shop scheduling and the like. These are so complicated that they require overconstrained linear programs which are heuristic in nature. The extent to which such programs depart from optimality can also be measured.

Three papers in the book concentrate on hardware; one example is where parallel and sequential processing are compared. The rest of the book concentrates on software—these vary over a whole range of matters such as complexity, strategy and stability in algebraic and other computational methods.

The book is well organised and physically well produced, presumably by a photographic method, and has a flexiback cover. It is rather specialised in its contents, but for those in the field it is a book that certainly deserves a place on the bookshelf.

F. H. GEORGE (Uxbridge)

Digital System Design Automation—Languages, Simulation and Data Base, by Melvin A. Breuer; 1977; 417 pages. (*Pitman*, £13·95)

This book is aimed primarily at programmers whose function is to provide software for computer hardware designers, and generally at anyone working near the hardware/software interface of digital systems.

Chapter 1, System Level Simulation, deals with conventional simulation languages. Implementation details are given for two such languages. The emphasis is mainly on the simulation of computer programs, and there is a large section on graph models of programs. Chapter 2, Register Transfer Languages and Their Translation, describes a register transfer language called DDL. In Chapter 3, Register Transfer Language Simulation, the problem of translating DDL descriptions into executable hardware simulation programs is tackled. A weakness of both these chapters is that too much emphasis is placed on minor details of a specific language; this tends to obscure more fundamental issues such as the problems of simulating parallel or asynchronous operations.

The fourth chapter, Design Automation Aids to Microprogramming, contains a good description of microprogramming, and illustrates the use of a register transfer language for describing the operating of a microprogram. The final chapter, Data Structures, Data Base and File Management, contains standard material on data base management and the representation of data.

The book has several weaknesses; in particular, it is excessively verbose. However there is a great shortage of literature on this subject, and the book fills an important gap.

PETER J. MOYLAN (Newcastle, Australia)