This discursive book, far more readible than anything previously available in its subject field, surveys compiler writing, touching in an introductory way on many principal compiler issues. It is quite suitable for classroom use in an introductory course, and also as a guide for the experienced programmer wishing to learn something of the inner workings of compilers. Its point of view is principally shaped by experience with FORTRAN compilers.

The layout of topics in this book is as follows. A first chapter discusses, in general terms, such basic terms as symbolic language, interpreter, compiler, bootstrapping, syntax-oriented translation. Chapter 2 introduces BNF as a mechanism for the definition of languages, and describes various possible additions to the basic BNF apparatus, that is, additions potentially useful in shortening syntactic descriptions. Chapter 3 is a broad introduction to the parsing problem, outlining top-down, bottom-up, and catch-as-catch-can approaches to compiling. After three additional chapters devoted to semantic issues, this discussion of parsing is continued in Chapter 7, which describes precedence parsing in its application to algebraic expressions and the use of precedence methods for translation from ordinary algebraic infix notation to Polish strings.

The remaining chapters of the book are concerned with the semantic portions of compilers, i.e., with the symbol table manipulating and code generating routines which compilers contain. Chapter 4 describes symbol tables in general, outlines the hash schemes by which they may be addressed, and surveys the lexical scan processes used to enter items into such tables. The same chapter goes on to discuss some of the basic object-code issues arising in the assignment of addresses to symbol table items: layout of arrays, analysis of equivalence declarations, treatment of COMMON blocks. Chapter 5 describes target code styles for the treatment of control statements, emphasising techniques available for use in "single-pass" compilers. Chapter 6 discusses some of the special issues arising in connection with FORMAT-controlled I/O statements, describing the structure of a FORMAT interpreter, and the way in which links between a program, its I/O subroutines, and an operating system may be constructed. Chapter 9 gives a general discussion of target code questions connected with subroutine linkages, indicating the manner in which these linkages may be compiled, describing the treatment of arrays when they occur as subroutine parameters, and discussing the special issues which arise when subroutine names are to be transmitted as parameters. Chapter 8 describes the code generation process in additional detail, indicating the manner in which straightforward code may be generated either from pre-compiled Polish strings or directly from algebraic formulae during a precedence parse and the manner in which simple local optimization may be included in this process. The same chapter also details code generation both for the addressing of indexed variables and for the invocation of separately compiled functions.

J. T. SCHWARTZ

Courant Institute of Mathematical Sciences
New York University
New York, New York 10012

18[12, 13.35].—F. GENUYS, Editor, *Programming Languages*, Academic Press, New York, 1968, x + 295 pp., 24 cm. Price $15.00.

This book contains a collection of five articles based upon a series of lectures given at the NATO Advanced Study Institute in 1966. The articles are all on a high technical level, though there is considerable variation in clarity of writing.

The first article is by C. C. Elgot, and is entitled "Abstract Algorithms and Diagram Closure." It is a difficult and highly mathematical paper concerned with the theory of computation. An abstract algorithm is recursively defined, starting with a class of elementary imperative sentences. These imperative sentences assume the existence of an abstract machine with cells capable of holding contents. An elementary imperative sentence causes the contents of certain cells to be examined, the contents of certain other cells to be replaced, and control to be transferred to some other imperative sentence. Each such sentence is defined by a certain mapping function, known as a *direction*. A network of these sentences is considered to be a diagram, and every diagram may itself be treated as a direction. Thus the closure $\overline{D}$ of a set of directions D consists of those directions that can be computed from diagrams composed of directions in D. The paper consists of an exploration of the concept of diagram closure and related computability results. My own, somewhat subjective appraisal is that this work is highly overspecialized and is primarily a mathematical exercise. However, one's reaction depends upon one's general attitude towards current work in theory of computation; and Elgot's article is in any case representative of the best work in the field.

The second article is by E. W. Dijkstra, and is concerned with cooperating sequential processes. This article was my favorite, and I consider the book worth buying for Dijkstra's article alone. It is a brilliant and clear exposition of the subtle difficulties inherent when sequential computing processes, operating in parallel with unknown relative speeds, must cooperate and share information. Dijkstra's basic synchronizing device is the *semaphore*, which can be operated on by two operations: the P-operation and the V-operation. The V-operation adds one to the value of the semaphore; the P-operation subtracts one from the value of the semaphore as soon as the resulting value would be nonnegative. Thus the V-operation is, roughly speaking, a go-ahead signal issued by one process to others; the P-operation is a wait-for-go-ahead. Dijkstra shows how semaphores can be applied to the management of input-output buffering, to communications systems, and to storage management. Although the article does not even assume that the reader knows ALGOL, it is nevertheless a highly sophisticated treatment of the problems of parallel processing.

The third article, "Compiler Writing Techniques" by L. Bolliet, occupies nearly half of the entire book. Although the article is not well written, it contains a great deal of useful information about compiling algorithms, particularly those concerned with syntactic analysis. Much of the article consists of listings of actual ALGOL programs (sparsely commented). The basic orientation is towards the construction of an interpreter (rather than a compiler) for ALGOL, though the difference is relevant only in the post-syntactic phases of a compiler. The last section of the article treats compilation for multi-access systems and the problems of incremental compilation.

The fourth article, "Record Handling," is by C. A. R. Hoare. A *record* is a computational entity used to represent an object that has several distinguished subparts. Objects with the same subparts may be lumped into a *record class*: the

subparts are known as *fields*. Since fields may themselves refer to other records, complex structures can thus be developed. Many ramifications of the record concept are explored, and the application of records in different programming languages is discussed. An appendix specifies the necessary additions to ALGOL 60 in order to include records. Although by this time Hoare's concepts are quite well known, the article is still well worth reading and helps to provide a unifying framework for a number of related approaches to record handling.

The final article, by Ole-Johan Dahl, is a survey of discrete event simulation languages. Considering that Dahl is one of the authors of SIMULA, a leading simulation language, this is a remarkably even-handed and impartial discussion of the field. The author considers five well-known simulation languages: GPSS, SIMSCRIPT, CSL, SOL, and SIMULA. Examples are drawn from all of them. Dahl discusses the peculiar requirements of simulation languages, and takes pains to warn the reader of the hazards of making predictions from computer simulations. This article is in much the same spirit as the Hoare article, and develops a unifying framework in which the different languages can be viewed. There are interesting interrelationships between simulation languages and Hoare's record concept, since simulated entities usually have just the kind of structure that Hoare is concerned with. However, since the articles are separately written, this connection unfortunately is not made explicit.

PAUL ABRAHAMS

Courant Institute of Mathematical Sciences
New York University
New York, New York 10012

19[13.35].—ROBERT E. LARSON, *State Increment Dynamic Programming*, American Elsevier Publishing Co., Inc., New York, New York, 1968, xvi + 256 pp., 24 cm. Price $14.50.

Of all the primary ideas in optimization theory, dynamic programming has perhaps the most immediate and intuitive appeal. So fecund of application is "the principle of optimality" that one can readily forgive its ignorance of the subjunctive for the sake of the constant challenge it presents to the ingenuity in adapting it to a vast range of situations. Its basic notion can be seen from the simplest of examples, yet it reaches to the most recondite problems. It is in fact—as I once heard Bellman remark—"just mathematics; not difficult; all it requires is intelligence."

The two main drawbacks of dynamic programming were its lack of precision and what Bellman colorfully called "the curse of dimensionality." The work of Berkovitz and Dreyfus in 1964 laid a rigorous foundation and permitted the proper conditions of continuity to be imposed on optimal solutions. At about the same time Larson was beginning to overcome the practical difficulties that arise with more than one or two state variables. It is this work that he has now most usefully presented in book form. State increment dynamic programming starts from the standard functional equation of dynamic programming, but uses two ingenious modifications to reduce the computing time and storage requirements. The first is to choose the time-like increment so as to keep each successive state within a prescribed hypercube centered on the previous one. The second is to carry out the