each legal position for the final correct algorithm. For comparison, the corresponding figures for the near-correct algorithm produced after six iterations to the automatic refinement process (excluding the 728 positions from which White fails to win) are given in parentheses. This algorithm also requires 60 ply to win in the worst case. In both cases, 67% of positions are played optimally, with no move increasing the depth by more than 13 ply.

From the previous discussion, it is clear that the method of win tree examination can enable a fully correct algorithm to be obtained by refinement within the overall framework of an original algorithm, as well as providing a means of testing the correctness of an algorithm without any knowledge of the total number of theoretical draws, losses or even legal moves.

By enabling the incorrect positions for a given algorithm to be examined and thus suggest possible modifications, this method would therefore seem to make the development of correct as opposed to optimal algorithms a feasible proposition.

**References**

Beal, D. F. (1977). Discriminating wins from draws in King and Pawn versus King chess endgames, Queen Mary College, London, Department of Computer Science and Statistics.

Bramer, M. A. (1975). Representation of knowledge for chess endgames, Mathematics Faculty Technical Report, The Open University.

Bramer, M. A. (1977). Representation of knowledge for chess endgames: towards a self-improving system, Ph.D thesis, The Open University.

Bramer, M. A. (1978). Computer-generated databases for the endgame in chess, Mathematics Faculty Technical Report, The Open University.

Bramer, M. A. (1980). An optimal algorithm for King and Pawn against King using pattern knowledge, in Clarke, M. R. B. (ed.), *Advances in Computer Chess 2*, Edinburgh University Press, pp. 82-96.

Bramer, M. A. and Clarke, M. R. B. (1979). A model for the representation of pattern-knowledge for the endgame in chess, *Int. J. Man-Machine Studies*, Vol. 11 No. 5, Sept. 1979.

Bratko, I. (1978). Proving correctness of strategies in the AL1 assertional language, *Information Processing Letters*, Vol. 7 No. 5, pp. 223-230.

Bratko, I., Kopec, D. and Michie, D. (1978). Pattern-based representation of chess endgame knowledge, *The Computer Journal*, Vol. 21 No. 2, pp. 149-153.

Huberman, B. J. (1968). A program to play chess endgames, Ph.D. dissertation, Technical Report no. CS 106, Stanford University, Computer Science Department.

Tan, S. T. (1972). Representation of knowledge for very simple pawn endings in chess, University of Edinburgh, School of Artificial Intelligence, Memorandum MIP-R-98.

Zuidema, C. (1974). Chess, how to program the exceptions? *Afdeling Informatica*, IW21/74, Math. Centrum, Amsterdam.

## Book reviews

*Introductory ALGOL-68 Programming*, by D. F. Brailsford and A. N. Walker, 1979; 281 pages. (*John Wiley*, £12·00, £5·95 paper)

Yet another textbook derived from a lecture course given to an undergraduate audience, this time at the University of Nottingham. The book is aimed at students learning their first programming language, and deals principally with the basic aspects of ALGOL-68 as may be applied to simple undergraduate problem solving activities. The authors argue that the more advanced features of the language are adequately dealt with elsewhere, and that even elementary ALGOL-68 provides a more efficient and attractive alternative to other languages now taught at this level. After a most welcoming foreword and introduction, Chapter 1 launches straight into the most crucial aspects of ALGOL-68 thinking, the concepts of objects and (reference) modes. This is immediately followed by a chapter on program structure, covering blocks, scopes, ranges, expressions, etc. Taken together they form the most formidable part of the book to the naïve reader (especially one with no ALGOL-60 or Pascal experience), with no running examples or problem solving goal in mind. However both chapters proceed at a slow pace, carefully explaining and expanding upon the issues.

It is only with Chapter 3 that we see the reason for it all; some elementary programs. Chapter 4 brings us arrays and structures followed by procedures in Chapter 5. Here we see the main weakness of the book; procedures are introduced too late, and their crucial importance is not emphasised sufficiently. The best that can be managed is that they are 'perhaps the cornerstone of good ALGOL programming'. Whether their value is obvious to the naïve programmer or not, I feel it does no harm to labour the point more than the authors have done here, particularly with a language of the pedigree of ALGOL-68. Chapter 6 follows with transput, and then there are four case studies, all with well stated problems and coded solutions in ALGOL-68. The final chapter deals briefly with advanced features, with particular emphasis on the dialectic differences. The appendices contain solutions to the excellent exercises, descriptions of the environments of revised ALGOL-68 and -68R, and a syntax chart of the language subset that the book describes (no syntax rules are stated formally in the text). There is also a very useful appendix on program development and error detection.

The book is written in a highly conversational style, with all the missionary zeal of self confessed disciples. Whether you like the jokes and asides or not, the book manages to convey better than any other I have read the underlying ethics of ALGOL programming; what is 'right' and what is not. Because this kind of background is so important this reason alone should commend the book to people for use with introductory programming courses or indeed to any wishing to absorb the ALGOL-68 style of programming.

Alan Blannin (Reading)

*Assemblers, Compilers and Program Translation* by P. Calingaert, 1979; 270 pages. (*Pitman*, £13·50)

Designed as a textbook for a one-semester course at the University of North Carolina, this book is about the mechanics of assembly, macroprocessing, compilation, modules and their activations, linkers, loaders, etc. It assumes the reader understands how to use program translators and wishes to study how they work.

There are eight chapters with plenty of examples and good reading lists. There are chapters on assembly, macroprocessing and program modules which are self-contained, but later chapters are not so well organised, although their content is satisfactory. Lexical analysis, syntax analysis (by recursive descent and by operator precedence methods) and hash tables are well explained, mainly by example. Semantic processing is treated, also mainly by example, but is not so clearly explained. Optimisation and code generation get slight treatment, as do linkers and loaders.

There is no theory in this book and little abstract discussion: the treatment is by explanation and example, which succeeds best in the early chapters, where the simpler translators are discussed. The style of writing aims at precision with some success but at the expense of succinctness.

A. E. Glennie (Reading)