

BOOK REVIEWS

LOGIC: A COMPUTER APPROACH, by Morton Schagrin, Randall Dipert, William Rapaport. New York: McGraw Hill, 1985, 320 pages.

It is amazing that virtually all logic texts ignore computer applications of FOL (first order logic). It is amazing, given the historic linkage between FOL and computer science, and given the continuing importance of mathematical logic in AI and other areas of Computer Science. Of course, there is BERTIE (and its descendents), a large CAI (computer assisted instruction) package which some universities have installed on their mainframes. But as yet the computer revolution has not greatly affected the teaching of logic, nor is it in evidence in most logic texts.

Messrs. Schagrin, Dipert and Rapaport have sought to correct things with their recent book *Logic: A Computer Approach*. They attempt to teach the basics of symbolic logic (i.e., sentential and quantificational calculus) by discussing algorithms to carry out various logical tasks (checking well-formedness, generating truth tables, checking natural deduction derivations, etc.). Most of those algorithms are standard in the literature, but it is very nice to see them drawn together in one text.

The authors have given us a text with a number of merits. The book is succinct, with good discussions of the history of attempts to mechanize reasoning, of Wang's algorithm, and subproof construction (pp. 168ff.). The exercises are numerous and good, as are the suggestions for implementing the various algorithms. Moreover, the appendices (one on switching theory, the other on Turing machines) are very appropriate.

However, the drawbacks of this text are numerous, at least from the perspective of the person who teaches FOL with a natural language orientation. For one thing, fully parenthesized notation is used throughout the book, whereas some informal rules regarding dropping parentheses could easily have been discussed and adopted. Also, some of the book's stylistic conventions are pedagogically cumbersome. For example, boldface letters are used as variables, which are difficult to write on the blackboard and too easily confused with constants. Again, the numerals '0' and '1' are used as truth values, which is mathematically nice but is too far divorced from natural language.

More troublesome is the short shrift given to topics so important in a first course in logic. As much time is spent on flowcharts as is spent on models. Worse yet, only 50 of the 340 pages are devoted to quantificational logic!

BOOK REVIEWS

Even from the point of view of computer science this text under review has drawbacks. The algorithms are in many cases very sketchily put (see, for example, p. 139), making implementation rather difficult for the average student. The algorithms given are almost all unstructured, with GOTO's and multiple halts everywhere. Amazingly, none of the algorithms are recursive — very unfortunate, given the role of recursion in modern programming. Thus, many of the algorithms are far more complicated than they need to be — compare, for example, the authors' algorithm for truth value calculation with the one found in Tannenbaum/Augenstein's *Data Structures Using Pascal* (Prentice-Hall). Flowcharts are used alongside PDL ("pseudocode"), resulting in considerable waste space. And the algorithms do not exploit the boolean functions present in virtually all higher-level languages. Finally, in the discussion of Wang's algorithm, the concept of a stack is invoked with little discussion of what a stack is and what the stack functions (push, pop, top, etc.) are.

As a matter of emphasis, I would suggest that topics such as normal forms, Polish notation and the connectives NAND, NOR, XOR be relegated to the appendices.

I suspect that the reason the book under review has the flaws mentioned above is that it attempts to do what is impossible. One simply cannot cover in one book that which is normally covered in a symbolic logic course (FOL including identity/definite description with a natural language focus) *and* what is normally covered in an introductory programming course. I would suggest that anyone attempting to design a computer-oriented logic text simply give up the hope of teaching computer program design, and instead focus on teaching logic, but have starred sections (which can be omitted by the computer illiterate) which would outline the relevant procedures. That way, those who want to learn just logic can do so; those (mathematics, engineering, and computer science students) who want to look at some sophisticated programs may do so. A CAI package could then be provided whose design is based upon the programs discussed in the book, and be made accessible to all students (whether or not they are accomplished programmers). Speculating further, such a CAI tutorial package would be designed best for microcomputers, since at most colleges it is easier for humanities students to have access to Apples than to mainframes.

We still have not seen a truly computer-oriented symbolic logic text.

Gary Jason

Departement of information and comuter Science
University of California, Irvine
Irvine, California 92717
USA