continuous algebras is exposited. In particular, an algebraic complete partial order $\mathbf{AT_S}$ of *strong acceptance trees* is defined. $\mathbf{AT_S}$ is shown to be the initial continuous algebra model for an extension of the axioms given in Chapter 2. The extension includes a process expression Ω for the divergent process. $\mathbf{AT_S}$ is intended as a model for the operational preorder $\sqsubseteq_{\text{MUST}}$; the chapter ends with a discussion of how it can be modified to get weak acceptance trees $\mathbf{AT_W}$ and acceptance trees $\mathbf{AT}$ appropriate for modeling $\sqsubseteq_{\text{MAY}}$ and $\sqsubseteq$ respectively.

Chapter 4 extends the calculus of process expressions by allowing recursive definitions. There is a discussion of the new syntax of process expressions, which now includes a concept of bound variable. A fixed-point semantics for recursively defined processes is given. After this, there is a substantial discussion of proof systems for the extended calculus: the addition of recursion necessitates the introduction of the induction principles to reason about recursively defined processes. The chapter ends by defining an operational semantics for recursive processes and proving full abstraction for $\sqsubseteq_{\text{MAY}}$, $\sqsubseteq_{\text{MUST}}$, and $\sqsubseteq$ with respect to $\mathbf{AT_W}$, $\mathbf{AT_S}$, and $\mathbf{AT}$ respectively. Again, this completes the demonstration of the desired relationships and concludes Part II.

Part III of the book consists of Chapter 5. In Chapter 5, two important new constructs are added to the syntax of process expressions: (1) a binary operator | for *parallel composition* of processes and (2) a collection of unary operators $\backslash c$ for *hiding* a process channel c. The chapter ends with a brief discussion of another form of communication principle popularized by C. A. R. Hoare. CARL A. GUNTER

ROB R. BRADY. ***The logicworks. Student manual.*** Philosophy Documentation Center, Bowling Green 1987, i + 21 pp. + 2 disks.

ROB R. BRADY. ***The logicworks. Guide for instructors.*** Philosophy Documentation Center, Bowling Green 1987, i + 23 pp. + disk.

Though many computer-assisted instruction tutorials are now under design, ***Logicworks*** is one of the few that are actually up and running, and on the market. It runs on the IBM PC (and clones) with two floppy disk drives. This reviewer has used the program in a number of courses, for both informal and formal logic, with both small and large enrollments.

Content. The exercises on informal logic cover a small number of elementary ideas: identifying the premisses and conclusions of an argument in English; critically evaluating a definition by judging whether it exhibits any of several possible defects; identifying informal fallacies; distinguishing emotive disagreements from those about truth; classifying sentences according to their illocutionary force into one of five categories; classifying argument forms from among a list of ten traditional types, like hypothetical syllogism; and evaluating syllogistic reasoning by Venn diagrams. Clearly the topics covered will not match what everyone wants to teach, but ***Logicworks*** covers more topics than most of the informal logic tutorials available.

The exercises and programming are not perfect. One tutorial, for example, asks students to distinguish premisses from the conclusion in an English text by highlighting each in a set manner. But the rules of highlighting are rather complicated and non-intuitive. Students, on the whole, had much more trouble learning how to highlight than how to distinguish premisses from the conclusion. It is unfortunate that this particularly frustrating exercise is the one that introduces students to the program package.

Most of the remaining tutorials in informal logic consist of classifying examples into one of several set categories. One difficulty with such problems is that the instructor may disagree with the answers programmed into the tutorial. This reviewer did so in a number of cases, and dealt with the problem by simply not assigning those exercises. An alternative is to edit the exercises (by using an editing program provided to instructors)—if the instructor is willing to deal with the extra complication of asking every student to make a copy of the edited exercise disk.

A general problem with informal logic is that its theoretical concepts are poorly defined, and definitional obscurity is exacerbated by the lack of uniformity across texts. The student handbook sold with ***Logicworks*** does not define the varieties of informal fallacy, definitional error, and illocutionary force presupposed by the programmer. Thus, prior to teaching the material, the instructor must engage in some tedious conceptual abstraction from the exercises to determine what notions the programmer had in mind. The package would be easier to teach and use if each of the categories sought in the answers was briefly defined.

One odd feature of the Venn diagram program is that instead of indicating that a region is non-empty by the usual convention of marking it with an X, the program calls for it to be colored in. Moreover, the

way to indicate that one of two regions is non-empty without declaring which one, is to color both. An ambiguity results: shading both A and B means that $A \cap B$ is non-empty and that $A \cup B$ is non-empty. Happily, due to the restricted set of syllogistic arguments in the exercises, no invalidity results. This particular exercise would be improved by using the ordinary conventions, by allowing students to design and check their own syllogisms, and by augmenting the syllogistic syntax so as to include singular propositions.

The material on formal logic teaches Copi's system. One set of exercises covers sentential logic and another first-order logic. For each there is one tutorial on translating from English into symbols, one on justifying proofs that are provided, and one on constructing proofs for set inferences. It is also possible for students to construct proofs for their own inferences and ask the program to check them. Apart from using Copi's system, the peculiarities of which are well known, the formal material is unobjectionable. A group of reluctant students was able to master Copi's system completely in several weeks working on their own, without complaining.

There should, however, be clearer definitions in the instruction book on the rules for indirect proof, conditional proof, and the quantifiers. One quirk of the proof programs is that some credit is given for every correct inference even if it does not lead to the desired conclusion. Some students just added and subtracted double negations for fifty lines, thus getting 99 points out of 100 without having a clue about the correct proof.

Administration. The program tries to combine the personal convenience of tutorials students can use at home with the monitoring features like grade collecting usually found in programs designed for a networked computer laboratory. It is true that the program succeeds in incorporating a workable grading system that collects scores for the instructor and obviates the need for correcting homework. Recovering scores on correct answers is accomplished by means of a feature built into each student's personal program disk that automatically keeps a hidden record of his or her responses. Periodically the instructor collects the students' disks, and disk by disk reads these hidden files into a special grade record by means of a "grading" program provided to the instructor. The claim that accompanies the program is in fact true: an assistant can collect, read, and return the disks, even for a large class (in one case 130 students) within a fifty-minute class period. The list summarizes the scores of all students, and it is possible to see or print out (given plenty of time) an individualized record for each student telling exactly what problems were answered correctly. It is also possible to enter test scores manually for tests that are not part of the computer tutorials and to have the various grades averaged according to assigned weights.

There is a trade-off, however, in trying to capture the advantages of mobility and grade collecting. It is that a fair amount of time has to be spent by the instructor dealing with students who have destroyed their disks, which they manage to do in the most creative ways. Be prepared to hear many sad stories.

If a disk was simply destroyed so it could not work, a student could just use his or her backup disk and there would be no great problem. But experience proves that in many cases the disk is only somewhat crippled by exposure to magnetic fields. It will still bootup and allow the student to try to answer some questions. It is at this point that problems surface. Either the program will not work or it appears to work but will not record the student's answers in the hidden file. In a number of cases the student exited the program and backed up the disk, erasing his previous backup in the process, only to discover that all record of his work was somehow lost or if not lost at least would no longer work with the student's program.

Another way in which students destroy their work is in learning to master how to back up a disk. Some students find this hard to do. The result is that they ignore the advice to back up their disks, or in attempting to do so manage to destroy their programs. This happens despite the fact that there is a simple backing up program provided. (In one case a teaching assistant even destroyed the instructor's class list file while collecting.) Students who try to configure the program for use with a hard drive have also destroyed their programs or grade record in the process. In this respect it would be useful if the program included a set-up routine for hard drives.

It is true that there are several special features built into the program to help the instructor in such cases. There is a second hidden score file kept on the students program disk, and an instructor can assign a new secret code number to a student so that his or her records from two program disks can be united into a single record. But correcting these problems takes time on the part of student and instructor. In more than half of the cases of this sort, this instructor found that the student had to buy a new disk and spend a fair amount of time redoing previous work. The expense and wasted time makes nobody happy. Given

human nature and the design of ***Logicworks***, the instructor must plan to spend a certain amount of time dealing with destroyed disks and designing humane grading policies. JOHN N. MARTIN

JON BARWISE and JOHN ETCHEMENDY. ***Turing's world.*** Kinko's Academic Courseware Exchange, Santa Barbara 1986, viii + 68 pp. + disk.

JON BARWISE and JOHN ETCHEMENDY. ***Tarski's world.*** Kinko's Academic Courseware Exchange, Santa Barbara 1987, vii + 85 pp. + disk.

If you have had a small amount of experience with a Macintosh and enjoy logical matters, then you will almost certainly enjoy using (or just playing with) these two wonderful pieces of software for the Mac. The reviewer understands that by the time this review appears, descendants of these two programs will be available from the Center for the Study of Language and Information at Stanford University.

Turing's world is a logician's dream come true: imagine being able to draw a Turing machine, design a tape, and then *see* the machine run on the tape. Now, with the aid of ***Turing's world***, you can actually do so by producing animated movies depicting the workings of Turing machines.

Here is how to run Turing. On opening the program, you see a blank space in which to draw a Turing machine (in the "node-arrow" format of, for example, Chapter 3 of Boolos and Jeffrey's ***Computability and logic***, XLII 585). To the left are a number of "buttons" with which, with the aid of the mouse and the keyboard, you draw and annotate the machine. First click on the top button (the node button). Next, lay down as many nodes (representing states of the machine) as you please in the drawing space by clicking at the positions in the drawing space at which you want them to appear. Click on one of two other buttons to choose the shape of arrow (more or less rounded) you want. Then, for each arrow that is to be part of the machine, click on the node that is to be at the tail of the arrow, and then on the node (possibly the same one, of course) that is to be at its head; as you do so, an arrow appears, connecting those nodes. When you have clicked on the head node, a window comes into view in which you choose a symbol in the machine's alphabet and an action (move left, move right, print one of the symbols). The arrow is now decorated with that symbol-action pair, and corresponds to a quadruple ⟨old state, symbol, action, new state⟩ in the machine's table. Repeat the construction of arrows until the machine is finished. Arrows and nodes may be deleted or dragged around the screen in the standard Mac manner. There are other buttons for inserting text and for redrawing and clearing the screen.

Below the drawing space is the machine's tape, a sequence of squares, all of them white, except for one "selected" black square in the middle. This square is surrounded by the scanning head, which looks rather like the cursor of a slide rule. Place the desired symbols on the desired squares of the tape (by clicking in an alphabet window and on the tape) and drag the head to the square on which the machine is to start.

Finally, select Go from the Execute menu (or type command-G) and watch the machine and the tape change. The node that represents the current state of the machine is black (the other nodes are white), and as the machine changes its state, the nodes representing the new and old states change colors, while the arrow connecting them shimmers slightly. The symbols on the tape and the position of the head change in synchronization with the machine's activity and in accordance with its design. The tape appears to be potentially infinite: when the head would otherwise go off the left or right end of the tape area, the tape and head jump back towards the middle, exposing more squares, and operation continues uninterruptedly. If and when the machine halts, the Mac's bell rings. The machine may be reset and the tape cleared by appropriate menu choices.

Some miscellaneous comments: Machines and tapes can be saved to, or read from, disk in the usual Mac way. ***Turing's world*** enables one to design and use *submachines*. By the time this review appears, an enhanced version of ***Turing's world*** will probably be available. The current version uses 90K. The program and manual are the best introduction to Turing machines the reviewer knows of, or can imagine.

Tarski's world (130K) may be of greater educational interest than Turing's. Its main purpose is to help students understand the notation of first-order logic. When the program is first opened, three windows are visible. The largest contains an 8×8 grid, seen in perspective. This is the window in which you design a possible world (model, state of affairs). To the left are buttons depicting a tetrahedron, a cube, and a dodecahedron. Clicking on one of these causes (one exemplar of) the corresponding solid to appear on one of the squares of the grid, from which it may be dragged to another square, or, in case of a mistake, off the edge of the grid altogether. One can change the sizes of solids on the grid, assign names ($a, b, \ldots$) to them, and alter the view of the grid from 3-D to 2-D. (Some solids may obscure others in 3-D view.)