The cellular logic array image processor

M. J. B. Duff and D. M. Watson

Department of Physics and Astronomy, University College London, Gower Street, London WC1E 6BT

Image processing and pattern recognition are tasks which are proving surprisingly resistant to automation. The majority of pattern recognition problems have still to be formulated in a manner sufficiently precise to allow effective programs to be written, whilst conventional approaches would seem to lead to a requirement for almost infinite processing times and storage area.

Conventional serial digital computers are not well matched to the processes involved in pattern recognition. A processor composed of an array of logic elements interconnected with nearest neighbours and processing in parallel is better matched and can operate at greater speed.

In order to explore fully the problems which are likely to be encountered in the use of parallel processing cellular logic arrays, several have been constructed and operated, the most recent of which, CLIP 3, is described.

(Received November 1974)

1. Introduction

Image processing and pattern recognition are tasks which are proving surprisingly resistant to automation. Fifteen or so years ago, there was considerable optimism about the possibility of constructing optically guided robots, reading machines, automatic optical inspection systems and similar devices in which the part played by human vision would be performed by clever combinations of optics and logic circuitry. This optimism was fed by an intuitive acceptance of the idea that the all-powerful computer could be programmed to solve any problem, given a large enough computer and a competent programmer.

Unfortunately, the majority of pattern recognition problems have still to be formulated in a sufficiently precise manner so as to allow effective programs to be written, whereas the 'steam-hammer' approach which is bound to work eventually (involving point by point comparisons between observed patterns and stored prototypes) would seem to require almost infinite processing times and storage area. At the same time, it has been humiliating to note that even the simplest of creatures is able to perform quite complex pattern recognition tasks without so much as a thought! Indeed, it is just this in-born ability to recognise patterns which makes it so incredible to the layman that the attempts to automate pattern recognition have been so ineffectual.

Conventional serial digital computers are not well matched to the processes involved in pattern recognition. Since such a computer can handle only one variable at a time, it follows that patterns have to be processed on a point by point basis, scanning the pattern as in a television raster. But many of the processes which one might wish to perform could, in principle, be executed simultaneously at all points in the pattern. For example, if we define edges of black figures in a pattern as being those black points which are adjacent to white points, then such points could be identified by examining all points in relationship to their immediate neighbours; and this is obviously a process which could, in principle, be carried out simultaneously over the whole pattern.

It is the purpose of this paper to suggest that a processor composed of an array of logic elements interconnected with nearest neighbours and receiving one input per element corresponding to the image density at the appropriate point in the pattern to be processed, is able to perform the majority of image processing operations in a manner which is well matched to the task to be performed (from the point of view of simplicity of programming) and at speeds far greater than those obtainable in serial computers.

The idea of using parallel processing techniques for pattern recognition is by no means new. Unger (1958; 1959) suggested a suitable structure for an array processor and McCormick (1963) developed ILLIAC III with a view to providing a system for analysis of bubble chamber photographs. Many other proposals have since appeared, but no large array optimised for image processing has yet been constructed. Usually, a serial hardware device is built and studies are then made as to the expected performance of an equivalent parallel hardware system. Preston (1971) and Kruse (1973) have proceeded in this manner. Earlier arrays—constructed by the authors have been described by Duff, Watson, Fountain and Shaw (1973).

Inevitably the processing speeds which might be achieved in serial systems are orders of magnitude slower that those which are obtained in parallel processing devices. Such large speed gains make feasible the use of process sequences which would be far too lengthy to merit serious consideration in serial systems.

To explore fully the problems which are likely to be encountered in the use of parallel processing cellular logic arrays, several have been constructed and operated, the most recent of which, CLIP 3 (Cellular Logic Image Processor 3) will now be described.

2. CLIP 3

Since there is to be one logic cell for each point which is to be resolved in the pattern, it is obvious that the complexity of the cell must be kept to the minimum consistent with the required processing capability, so as to avoid excessive costs.

The logic of each CLIP 3 cell is shown schematically in **Fig. 1.** A hundred and ninety-two cells are formed into a 16 by 12 cell square array, (in which each cell is connected to its immediate ring of eight neighbours by means of the interconnecting output and input lines shown in the figure). A is one bit of a 192 bit shift register, connected with similar bits in adjacent cells so that an image can be entered into the array as a serial string of binary digits. B is a second shift register which can contain another image formed during processing. Both registers can be cycled to display their contents as dot patterns on a two beam oscilloscope (see, for example, **Fig. 2**).

'1' cells are displayed as bright dots corresponding to white cells and '0' cells are displayed as fainter dots corresponding to black cells. The operation of the array is as follows: at $t = 0$ all control lines (G_1 to G_8, t_1 to t_3, C_1 to C_8) are set at zero and there is no signal flow in the system. At the first clock pulse, an appropriate selection of control lines are set at 1 and each Boolean Processor forms an output D as a Boolean function of

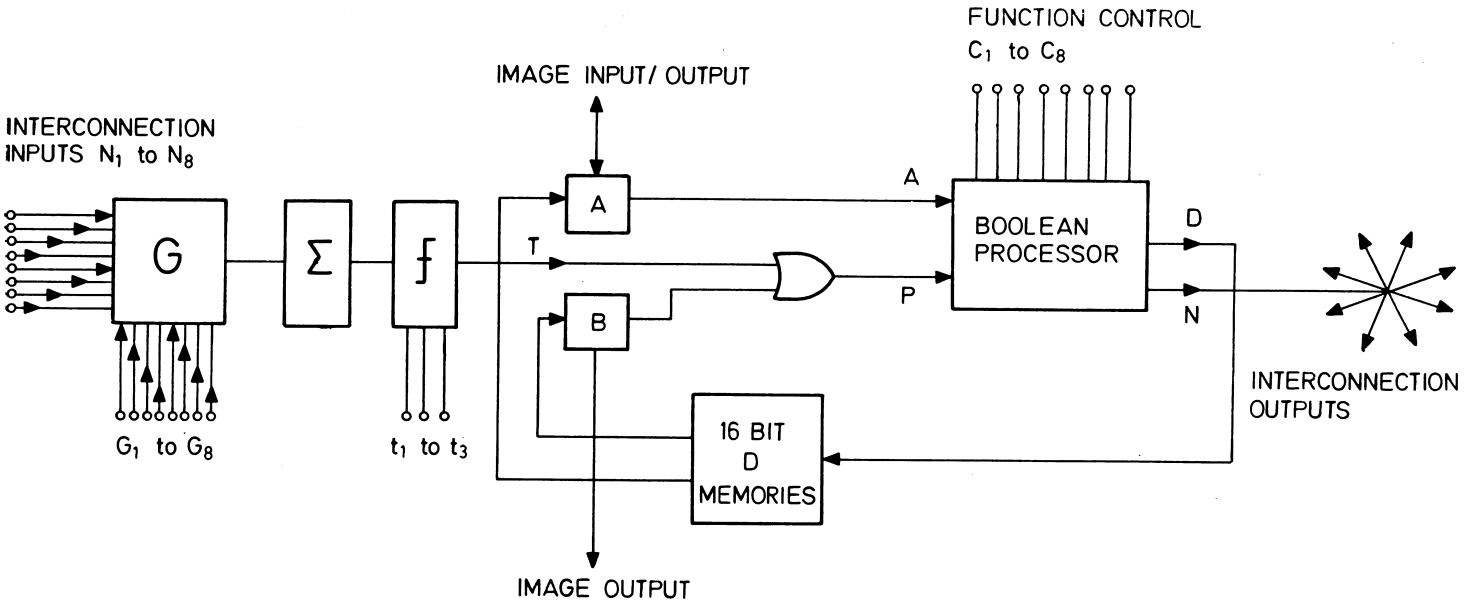


Fig. 1 Logic diagram for the CLIP 3 cell

A and P, and an output N as another Boolean function of A and P. The flow to neighbouring cells of the interconnecting output N is regulated by the gates selected by control lines G_1 to G_8 Each cell sums and thresholds the selected interconnection inputs (t_1 to t_3 determining the threshold level), forming a binary output T. If a pattern is stored in B, then each bit of the pattern OR's with the corresponding threshold output T to form the P input to the Boolean processor. After a suitable time interval has elapsed (usually about 3 μ seconds), the interconnection signal flow N reaches a steady state in every cell so that the final values of A and P are established throughout the array; at this point, the D output is clocked into one of the 16 levels of the D memories. For future processing, the original pattern can again be read into A or both A and B can be loaded from patterns stored at various D addresses.

In understanding the mode of operation of the CLIP 3 processor, it is helpful to see each process as being in two stages; the first stage establishes an interconnection flow pattern and the second stage uses the established flow inputs P together with the pattern input A to derive the new pattern D. There are three possible sources for the interconnection signal. These are:

1. The pattern in A; the Boolean function in the processor can be chosen so that the interconnection signal N is a function of A or its complement.
2. The pattern in B; since P is the OR of B and T, any 1 bits in B will inject an interconnection signal into the corresponding cell.
3. The edges of the array; for cells on the array margins all interconnection inputs N_1 to N_8 which would connect with cells outside the array are connected to a common bus. If this bus is itself connected to a 1 potential, then it will provide an interconnection signal source.

Once the signal sources have been established, and the Boolean function relating N to A and P and the threshold level selected, the interconnection signal can be traced through all the accessible cells in the array (taking into account the direction gates G_1 to G_8). As an example, consider the following set of conditions:

Array Edge	= 1
Threshold	= 0
Gates	= All open
B register	= All zero
N	$= P.\bar{A}$
D	$= P.A$

The Boolean function for N implies that black 0 cells receiving an interconnection signal will transmit it. The source of the signal is the array edge and all direction gates are open with a zero threshold; thus provided a black cell receives a signal from at least one neighbouring cell, it will pass it through its threshold gate. After a short time, all black 'background' cells will have a P input equal to 1. Also, white cells adjacent to black background cells will also have P equal to 1. The Boolean function for D is then applied and D will be 1 only for white cells with P equal to 1, i.e. for white cells on the outer edges of white objects. The output pattern D will therefore consist of only those cells. On changing the Boolean function for D to

$$D = \bar{P}.\bar{A}$$

the output pattern becomes black cells which are not in contact with the black background cells. These are black cells contained within closed loops of white cells, i.e. black holes in white figures on a black background. Examples of complex sequences of instructions will be given later after the description of the programming language.

3. CLIP 3 programming language

CLIP 3 is controlled by sequences of 24 bit instruction words; up to 256 words can be stored in a random access memory which is loaded either by keying in the instructions or by means of punched tape. Similarly, programs can be output onto punched tape.

A mnemonic language has been developed for CLIP 3. Experienced operators find no difficulty in translating this into the 24 bit code during loading; also a PDP11 assembler has been written which works in conjunction with the text editor to produce a CLIP 3 binary tape from a mnemonic source program.

Three types of instruction are available: LOAD, PROCESS and BRANCH.

1. *LOAD instructions*

LD a, b, d, S

a = D address from which register A is loaded

b = D address from which register B is loaded

d = D address into which the processed pattern is loaded

S is present when the array is in square format. If S is omitted, the array interconnections are automatically modified into hexagonal format.

If A is written instead of a number in the a field, then the contents of the A register are left unchanged (similarly for B in the b field). A C in either field clears the appropriate register. A D placed after the address in the a or b fields cycles the appropriate register and displays its contents as a dot pattern on a display oscilloscope.

2. *PROCESS instructions*

$$D = t(G)B_N, B_D, ES$$

t = threshold level (0 to 7). The threshold gate transmits a 1 when this threshold is exceeded.

G = list of active gate directions

B_N = any Boolean function of P and A, determining N

B_D = any Boolean function of P and A, determining D

E is present when the array edge bus is set at 1 and omitted when set at zero

S indicates a square format as in the LOAD instruction.

The processed pattern produced by the above function will replace the existing pattern in the D address selected in the LOAD instruction. Alternatively, by writing:

$$D = D.(\text{function})$$

or

$$D = D + (\text{function}) ,$$

the new pattern can be combined with the existing pattern. The instruction:

$$D \neq (\text{function})$$

implies that the processed pattern is formed but not loaded into a D address. Instead, a one bit register R is set equal to the output of a large AND gate with inputs from each D output in the array. Thus if R equals 1, it is implied that the output pattern D contains only 1 cells.

The functions may be written in the simplest self consistent form which is unambiguous. Thus

$$D = P.A$$

is allowable and implies that no interconnections are active. In this case, P and B are identical in each cell.

3. *BRANCH instructions*

(*a*) *BR C, R*, 3210, *I*

(*b*) *BTS C, R* 3210, *I*

(*c*) *BFS C, R*, 3210

C = 1 or 0, indicating that the branch will or will not be taken if at least one of the following conditions is met

R is present if the state of the array AND gate register is to be one of the branch conditions.

The selection of 3, 2, 1 and 0 indicates which of the control panel switches S_3 to S_0 are to be inspected as further branch conditions.

I = branch destination address

BR = normal branch to address I

BTS = branch to a subroutine at address I

BFS = branch from subroutine re-entering the main program at the address following the *BTS* instruction

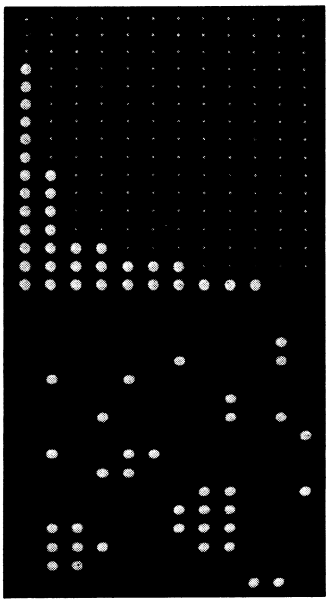
Subroutine branches nest up to 16 levels.

Example:

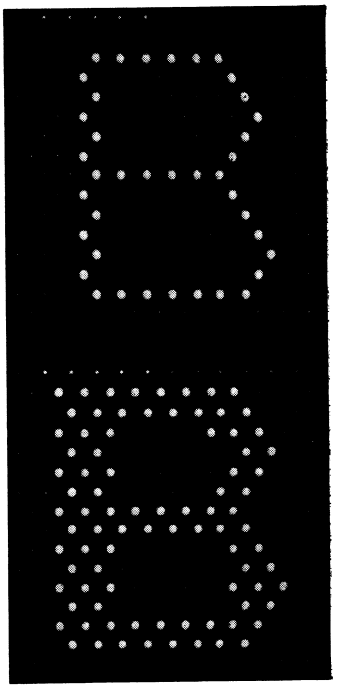
32 *BTS* 0, R, 3 50

Instruction 32 causes a branch to a subroutine commencing at instruction 50 provided neither the R register nor switch $S3$ is set at 1.

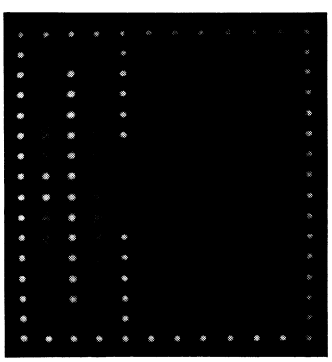
Various other operations are available in the control system,



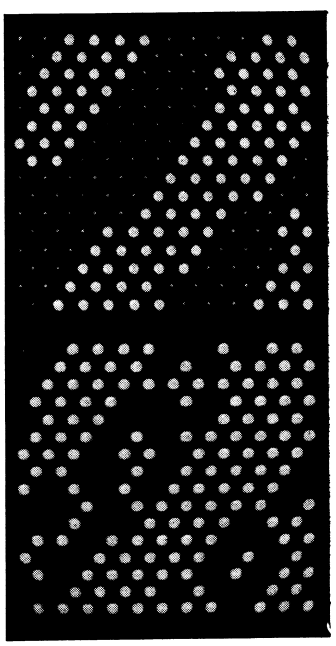
(*a*) **Object analysis program: objects in the lower display are sized and an areas integrated histogram plotted in the upper display**



(*b*) **Skeleton program: the upper display shows the unit width line skeleton of the figure below**



(*c*) **Laplacian program: a composite two beam, grey level display showing a voltage field between electrodes (see text)**



(*d*) (right) **Spatial frequency filtering: high spatial frequency components of the lower display (i.e. small details) are shown removed in the upper display**

Fig. 2

such as manual jumps to instruction addresses. One important operation is pattern loading. Patterns are loaded into the A register by cycling the register and using a light pen on the display screen. At each cycle, the cell value corresponding to the position of the pen on the dot display can be set at 1 or 0 ('WRITE' or 'ERASE' modes). A program loop of the form:

```
0  LD 0D, C, 0
1  D = A
   ⋮
I  BR 0
```

loads a pattern into the D_0 memory. Patterns can also be read into A from punched tape.

As an example, the following program will accept patterns loaded from the light pen and will find and display the outer edges of black objects on a white field. The A and B dot displays

appear on the same screen, one above the other (see Fig. 2). The lower (A) display is the input pattern and the upper (B) display the processed pattern.

0 LD 0D, 1D, 0, S	1 μ seconds
1 D = A, S	1 μ seconds
2 LD 0, C, 1, S	1 μ seconds
3 D = 0(1 → 8) PĀ, P.A, ES	3 μ seconds
4 BR 0	1·2 μ seconds
Total process time	7·2 μ seconds

The total process time is also shown. In fact, the 1 μ seconds time for instruction 0 is a nominal allowance for a non-display load instruction. A display instruction actually takes 6 ms, being limited by the oscilloscope and light pen characteristics.

In this second example, an object is displayed in the output when it is the only object in the input. In other words, an output display is obtained only when all non-zero parts of the input image are connected.

0 LD 0D, 1D, 0, S	Load input into $D0$ using the light pen and display
1 D = A, S	final output $D1$ after branching back.
2 LD 0, C, 1, S	Expand all objects sideways and downwards into
3 D = 0(2, 4, 8) P + A, P + A, S	$D1$.
4 LD 1, C, 1, S	Detect the top left hand corner point of the expan-
5 D = 0(2, 8)A, P̄A, S	ded objects and enter into $D1$.
6 LD 0, 1, 1, S	Propagate in $D0$ from the detected point in $D1$, and
7 D = 0(8) PĀ, PA, S	hence find the top left hand point of an object in $D0$. Enter this point in $D1$.
10 LD 0, 1, 1, S	Propagate from this point through all connected
11 D = 0(1 → 8) PA, PA, S	non-zero points in $D0$ and enter in $D1$.
12 D ≠ 0(1 → 8) P̄A, PA, S	Test for absence of non-zero points unconnected
13 BR 1, R, 0	to first object and branch to the display/program start.
14 LD C, C, 1, S	If more than one object present, clear the output
15 D = A, S	array and branch to the
16 BR 0	display/program start.

(The instructions are numbered octally. Note that the program will always branch back to the start at instruction 0, either from instruction 13 or from instruction 16. The program forms a single closed loop).

4. Some CLIP 3 programs

An important consideration which has already been mentioned is whether or not the chosen logic structure of the cells in the array is sufficiently powerful and general to allow economic solutions of image processing and pattern recognition problems. It should be remembered that some parts of these solutions will be processes which are more serial than parallel in concept; for example, counting. Will the array prove to be efficient in handling such tasks? The following examples illustrate some of the studies which have been carried out.

1. *Parameter extraction program*

The input array contains a pattern containing any number of objects (connected regions of 1 cells in a background field of 0's). The output 'pattern' is a histogram which can take various forms depending on the state of control switches S_0 to S_3. The histogram can display OBJECT AREAS, OBJECT PERIMETERS, OBJECT HORIZONTAL PROJECTIONS; the horizontal scale (object 'size') can be LINEAR or LOGARITHMIC and the vertical scale (object frequency) can be LINEAR or BINARY; the histogram can be NORMAL or INTEGRATED (frequencies of objects with parameters ⩾ the displayed variable). The number of CLIP instructions in this program is 70 and execution times vary between 23 μ seconds and 8,100 μ seconds.

2. *Skeleton program*

The input array contains an object whose shape can be represented by the central line in the thick branches of the object. The program thins the object to the central line and maintains connectivity (i.e. does not fragment the skeleton). The program comprises 65 CLIP instructions and a typical execution time is 270 μ seconds. The algorithm is one of several which have been developed and described in the image processing literature. The version illustrated is due to Watson (1974).

3. *Laplace program*

Using a light pen, a zero potential electrode structure is drawn on the input array display (the array boundaries are set automatically to zero). The program is stepped into the next section and electrodes at a fixed potential (normally 31 volts) are drawn. The program is stepped again and the output array displays the resultant potential distribution coded as a spectrum of grey levels. In effect, grey levels representing the starting voltages are 'smeared out' by an averaging proceess, thus calculating the potential field at every point in the array. The process involves replacing each point by the average of its four neighbours and iterating to a convergent solution. Iterative procedures for solving the Laplace equation are discussed by Baden Fuller (1973). 241 CLIP instructions are required and typical solutions converge in a few milliseconds.

4. *Spatial frequency filtering*

The input array contains the pattern to be processed. The output array contains the processed pattern. Operations involved include SHRINKING (in which white objects are shrunk by surrounding all black cells by one layer of black cells, thus stripping off the edge cells of the white objects), EXPANDING (the inverse of shrinking) and BOOLEAN functions combining various states of the pattern before, during and after shrinking or expanding. In this way, picture details of any selected magnitude can be eliminated or retained. Since each operation involves only two instructions, typical program lengths are, at the most, a few tens of instructions and execution times a few tens of microseconds.

5. Future developments

Although an array as small as CLIP 3 is unsuitable for 'production' image processing, it is quite adequate for testing algorithms and for studying optimisation of cell logic structures. CLIP 4 will embody the final optimised logic in an LSI chip (probably 8 cells/chip) and it is planned to have in operation by early 1977 a 96 cell by 96 cell LSI array which will be used for a range of image processing applications, particularly in the biomedical field. As a preliminary exercise, CLIP 3 is being scanned across a 96 × 96 point television image, using special purpose interactive hardware and a range of input optical peripherals (projection microscope, overhead projector, etc.) in conjunction with the television camera. CLIP 4 will be slower than CLIP 3 but it is confidently predicted that there will still be several orders of magnitude speed gain on any serial system.

6. Acknowledgements

The CLIP project is supported by the Science Research Council and by the Department of Physics and Astronomy at University College London. The authors are grateful to the technical staff of the Image Processing Group who have collaborated in this project (particularly to T. J. Fountain, M. Postranecky and G. K. Shaw).

References

BADEN FULLER, A. J. (1973). *Engineering Field Theory*, Pergamon Press.

DUFF, M. J. B., WATSON, D. M., FOUNTAIN, T. J. and SHAW, G. K. (1973). A cellular logic array for image processing, *Pattern Recognition*, Vol. 5, pp. 229-247.

KRUSE, B. (1973). A parallel picture processing machine, *Trans. IEEE*, Vol. C-22, No. 12, p. 1075.

MCCORMICK, B. H. (1963). The Illinois pattern recognition computer-ILLIAC III, *Trans. IEEE*, Vol. EC-12, No. 6, p. 791.

PRESTON, K. Jr. (1971). Feature extraction by Golay hexagonal pattern transforms, *Trans. IEEE*, Vol. C-20, p. 1007.

UNGER, S. H. (1958). A computer orientated toward spatial problems, *Proc. IRE*, Vol. 46, No. 10, p. 1744.

UNGER, S. H. (1959). Pattern detection and recognition, *Proc. IRE*, Vol. 47, No. 10, p. 1737.

WATSON, D. M. (1974). The application of cellular logic to image processing, Ph.D. Thesis, University of London.

Book review

Programs, Machines and Computation, by K. L. Clark and D. F. Cowell, 1976; (*McGraw-Hill*)

Theoretical options are not generally popular with computer science students. Various reasons can be adduced to explain this. Firstly theoretical courses are considered to be more difficult than nontheoretical courses. Secondly they often appear to have little relevance in real situations, and finally what relevance they do have is often obscured by vast amounts of meaningless definitions and symbolic notation. I do not believe that it is possible to avoid the first objection—a good theory should have predictive as well as explanatory qualities and will inevitably be more difficult to understand than an empirically observed fact. However textbooks on the theory of computer science should always attempt to avoid the second and third criticisms.

The book by K. L. Clark and D. F. Cowell can be split into two parts—the first four chapters in which the material is well motivated and well presented, and the final three chapters in which the authors succumb to illusions of mathematical elegance and, consequently, lose all sight of the practical relevance of their subject.

The first three chapters lead up to the standard results on the undecidability of the halting problem of Turing machines (presented in chapter four) and are concerned with demonstrating the Church-Turing thesis—that any computable function is computable on a Turing machine. The choice of material here would appear to owe a lot to Minsky's book *Computation: Finite and Infinite Machines* and includes proofs of the equivalence of Turing machines with unlimited register machines and recursive functions. Yet the order and manner of presentation improves, I feel, on Minsky's and should appeal to computer science students. One feature of the book, which is very much to be applauded, is the addition of assertions establishing 'correctness' to all their programs. Exercises, at this stage of the book, are also well-chosen to illustrate and add to the material in the text.

Chapter 5, entitled 'Machines with input and output streams' is, in my opinion, the worst of the book. It is this chapter which the authors would probably regard as the most novel. So far as I can discern the main result in this chapter is that the set of all strings generable by a class of (deterministic) machines is identical to the set of strings accepted by the corresponding class of nondeterministic machines. However this result is obscured by the authors efforts to accommodate the more usual definitions of pushdown automata and finite-state machines into the formalism which they established in chapters 1 to 4. This predilection for mathematical elegance is continued in chapters 6 and 7 where proofs of a number of solvability results are chosen for their brevity and aesthetic (?) value rather than their practical value. For instance to test whether a finite-state machine having m modes accepts the empty set we are told to try all inputs of length $\leq m$! Similar algorithms are given for constructing a deterministic finite state machine from a nondeterministic one, testing two regular languages for equality and testing whether a context free grammar generates the empty set The combined effect of all these brief but impractical proofs is that when the authors prove an important result which the students can relate to their own experience—that the equivalence problem for (monadic) program schemas is solvable—the authors are unable to provide exercises illustrating the result, the simple reason being that it would take a very patient and careful student a very long time to follow through the steps in their proof on even a small example. In fact all the problems referred to above can be solved using variants of a simple graph searching technique. Had the authors included a short section on graph searching and then used the technique in their proofs it would have improved the last three chapters considerably.

On the whole the book contains few typographical errors or errors of fact. I shall be communicating errors I have found to the authors but the following more major errors are worth noting. An error on the last line of page 31 is amusing. The line reads

$$^{R}x_i \leftarrow x_j\colon \{n_i\}_{i=1}^{\infty} \mid \rightarrow \{n_i'\}_{i=1}^{\infty} \text{ where } n_i' = n_j$$
$$\text{and for all } i \neq j,\ n_i' = n_i\ .$$

There is clearly a typographical error here since the final j should in fact be i. Correcting this we obtain

$$^{R}x_i \leftarrow x_j\colon \{n_i\}_{i=1}^{\infty} \mid \rightarrow \{n_i'\}_{i=1}^{\infty} \text{ where } n_i' = n_j$$
$$\text{and for all } i \neq i,\ n_i' = n_i\ .$$

What a howler! Clearly there is a clash between the free variable i and the bound variable i. Clearly also the authors never expected anyone to read this line of text! The proof of theorem 1.6.3 is false—a proof must surely involve a graph searching technique (yet another reason for including graph searching in the text). In addition the proof of theorem 2.2.6 is incomplete—the authors prove a sufficient condition for one machine to simulate another, but by the time they reach theorem 2.2.6 they seem to have forgotten the existence of the condition. In fact their proof is correct but it assumes a less stringent notion of 'stepwise simulation'. Finally on page 52 they suggest methods of simulating $x \leftarrow x + 1$ and $y \leftarrow y + 1$ on a Turing machine which are unnecessarily complicated—as was pointed out to me by one of my students in the middle of a lecture!

In spite of these criticisms I feel that the book can be recommended to computer science students. The first four chapters cover basic material on unsolvable problems which should form a compulsory element of all computer science degrees. The material is extremely well presented and indeed, in my opinion, better presented than in any other text I know. The final three chapters mainly cover 'optional' material and hence one's appreciation of the choice of material will be very much more subjective. My criticisms reflect my own views on the importance of the relevance of theory, but it may be that others will find the last half of the book as much to their liking as the first.

R. C. BACKHOUSE (Edinburgh)