Correspondence

To the Editor
The Computer Journal

Sir,

In a recent paper Williams and Ossher (1978) state that unstructuredness in flow diagrams is due to the presence of one or more of five constructs. We have suggested an alternative approach in characterising unstructuredness based on the premise that poor structure results from irreducibility (Cowell, Gillies and Kaposi, 1978; Gillies, Cowell and Kaposi, 1978; Kosaraju, 1974). (Briefly, a flowchart is irreducible if its flowgraph has no proper subgraphs with only one exit). By this definition, only 1(b) and 1(e) of the authors' constructs are irreducible. For example it will be noted that Figure 1(d) of (Williams and Ossher, 1978) is reducible, and by the technique of unfolding may be transformed into a pair of nested **while** loops:

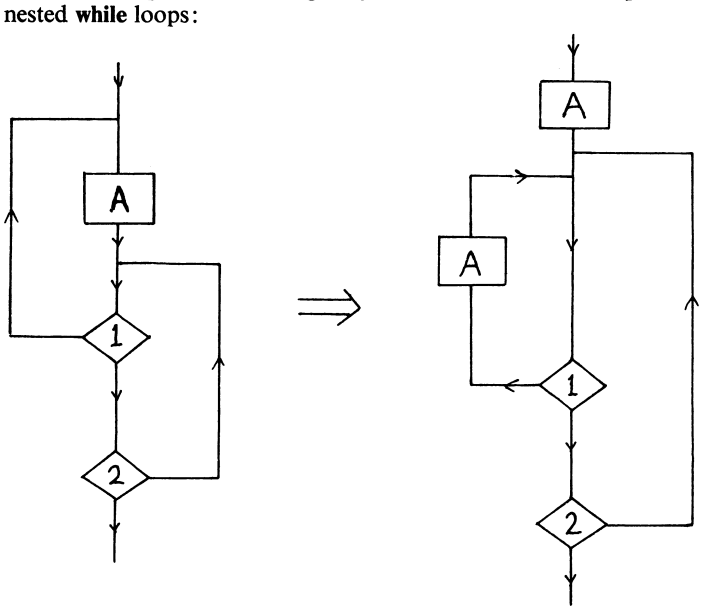

Figure 1(*d*)

Similarly, Figure 1(*a*) unfolds into two nested **if-then-else** structures and 1(*c*) into an **if-then-else** followed by a while loop.

The authors' algorithm when presented with a reducible construct, produces a structured equivalent containing two extra state variables, (see Figure 7(*a*) and (*b*).) This increases the cyclomatic complexity and hence may make the program harder to understand.

The authors' Figure 1(*b*) and (*e*) are both examples of irreducible forms. 1(*b*) is an instance of the only irreducible form with two tests but there are altogether six with three tests, including the one identified in Figure 1(*e*). There are 54 irreducible forms with 4 tests and 734 with 5 tests. It is possible to identify irreducible components of a flowchart using relatively straightforward algorithms (Gillies, Cowell and Kaposi, 1978), and if desired, to remove each irreducible component using techniques such as those described in this article, and elsewhere (Knuth and Floyd 1971 and 1972).

Yours faithfully,

A. A. KAPOSI D. F. GILLIES and D. F. COWELL*

Department of Electrical and Electronic Engineering
Polytechnic of the South Bank
Borough Road
London SE1 OAA

*Thames Polytechnic
12 September 1978

References

COWELL GILLIES and KAPOSI. (1978). Introduction to Flowgraph Schemas *Proc. CISS*, March 1978, Johns Hopkins University, Baltimore, USA.

GILLIES, COWELL and KAPOSI. (1978). Theory of Flowgraph Schemas *Proc. CISS, Ibid.*

KNUTH and FLOYD. (1971). Notes on Avoiding 'Go To' statements, *Info. Proc. Letters*, Vol. 1, Errata (1972), p. 177.

KOSARAJU. Analysis of Structured Programs, *JCSS*, Vol. 9, pp. 232-255.

WILLIAMS and OSSHER. (1978). Conversion of unstructured flow diagrams to structured form. *The Computer Journal*, Vol. 21, No. 2.

To the Editor
The Computer Journal

Sir,

We, the authors of the book, *Syntactic Pattern Recognition Applications* (Springer-Verlag, 1977) have found a very dubious book review published in the February 1978 issue of *The Computer Journal*. We have found the review to be biased and misleading, apparently written by a reviewer who is not aware of the recent progress in this field. The fact is that in addition to the recent successful applications of syntactic pattern recognition to waveform analysis and spoken word recognition, there are actually machines (not just computer programs) designed and built for automatic inspection (e.g. at GE and Philips) using the syntactic pattern recognition approach. These all happened during the last three or four years. In addition, the syntactic approach is quite compatible with artificial intelligence methods for 'understanding' patterns; rule based systems are in common use for this purpose, with the possibility of dynamically loading rules to handle specific contexts and situations. The syntactic approach is certainly more than eight years old, but syntactic pattern recognition, particularly its applications described in the book, has advanced significantly in recent years.

A brief check of the references in the book would have revealed to the reviewers that Chapter 2 is based on work published in 1975, Chapter 3 is based on work published in 1975, Chapter 4 is based on work published in 1975 and 1976, Chapter 5 is based on work published in 1972 and 1976, Chapter 6 is based on work published in 1975, Chapter 7 reports recent work never well documented elsewhere, Chapter 8 is based on work published in 1975 and 1976, Chapter 9 is based on work published in 1976, and Chapter 10 is based on work published in 1975. The work reported in this book has been published in such accepted publications as: Proceedings of the International Joint Conference on Artificial Intelligence, Proceedings of the International Joint Conference on Pattern Recognition, *IEEE Transactions on Computers*, *IEEE Transactions on System, Man and Cybernetics*, *IEEE Transactions on Acoustics, Speech and Signal Processing*, *Communications of ACM*, and *Computer Graphics and Image Processing*.

There is no doubt that syntactic pattern recognition has its limitations and weaknesses, as do the subjects of many other books that survey the state-of-the art, such as in artificial intelligence, statistical pattern recognition, etc. The reviewer certainly has his right to state his difference of opinion about a particular approach. However, it is irresponsible to the scientific, particularly to the pattern recognition community, for a reviewer to simply ignore all the facts and produce a review based on his biased opinion. We challenge the reviewer to produce evidence and facts from the past publications to support his statements

The first three paragraphs of Chapter 3 were an abstract which was

mistakenly included in the text The author indicated that these were to be deleted on the galley proofs but to our embarrassment they made it into print. The transformation from Clowes to Cloves is probably due to translation from English to Hungarian and the retranslation unfortunately did not produce the unique inverse. We appreciate very much the careful detection of these from the reviewer.

In summary, we feel that it is truly unfortunate to have such an inaccurate and biased review published in your prestigious journal and hope that this note will be used to correct such an error.

Sincerely,

J. ALBUS
R. H. ANDERSON
J. M. BRAYER
R. DEMORI
K. S. FU
S. L. HOROWITZ
B. MOAYER
T. PAVLIDIS
W. STALLINGS
T. VAMOS

School of Electrical Engineering
Purdue University
West Lafayette
Indiana 47907
USA
26 September 1978

To the Editor
The Computer Journal

My paper 'Generating permutations by choosing' appeared in the last issue of this journal (vol. 21 no. 4). In it I presented, as an application of one of the particular permutation generators discussed, a solution to the *n-queens problem* which is faster than the standard solution derived in Wirth's *Algorithms + Data Structures = Programs*. The program is correct; but unfortunately is less efficient than it should be.

In the text I stated (correctly) that 'since there can be only one queen per column, a solution (if one exists) consists of a permutation of the numbers 1 to n'. The whole point is that the use of permutations ensures that no two queens are placed on the same row or same column. Thus the **Boolean array** *col* as used in Wirth's algorithm is redundant, and should have been deleted from the subsequent text and from the procedure of Fig. 6. A revised version is attached. It is more elegant, less asymmetric and faster than the original.

I offer my apologies to your readers for my incompetence.

Yours sincerely,

J. S. ROHL

The University of Western Australia
Department of Computer Science
Nedlands, WA 6009
20 December 1978

```
procedure solve queens problem (n);
value n; integer n;
 begin
 integer array p[1:n];
 Boolean array upl[1-n:n-1], upr[2:2*n];
 procedure choose (k);
 value k; integer k;
  begin
  integer temp, i, pk;
  temp := p[k];
  for i := k step 1 until n do
   begin
   pk := p[i];
   if upl[k - pk] ∧ upr[k + pk] then
    begin
    upl[k - pk] := upr[k + pk] := false;
    p[k] := pk;
    p[i] := temp;
    if k ≠ n then choose (k + 1)
    else solution available in p;
    p[i] := p[k];
    upl[k - pk] := upr[k + pk] := true
    end
   end;
  p[k] := temp
  end of procedure "choose";
 integer i;
 for i := 1 step 1 until n do p[i] := i;
 for i := 1 - n step 1 until n - 1 do upl[i] := true;
 for i := 2 step 1 until 2*n do upr[i] := true;
 choose (1)
 end of procedure "solve queens problem";
```

Fig. 6 A procedure for solving the *n-queens problem*