To the Editor
The Computer Journal

Sir,

We have noticed with interest the recent articles on the investigation of parallel process by using state transition rules (Llewellyn, 1973; Lister, 1974).

Unfortunately although these methods provide elegant theoretical solutions, they fail to provide a satisfactory practical answer because of the 'combinatorial explosion' of states as the number of processes increases. It also seems to be the case that although a small system of correctly intercommunicating processes may easily be checked, minor errors can lead to such a profusion of states as to fill the available core.

An example which illustrates the point has been taken from one of our current programs for EEG analysis which runs on an Elliott 905 computer using a message passing scheduler (Townsend, 1972).

Our system restricts the interaction of processes to passing 'messages' from one to another via single-buffered 'routes'. A message in this context is simply a unit of information, and the whole mechanism can be represented by a simple non-deterministic model as follows:

The possible states of a process are numbered 0 to N and transitions between states occur only when a message is picked up from a route (operation GET) or placed on a route (operation PUT).

Allowable transitions are specified by statements of the form

$$[\text{TRANSITION}] = [\text{OPERATION}][\text{ROUTE NO}][\text{STATE NO}] \rightarrow [\text{STATENO}]$$

A complete process is specified by a number of such statements and the whole system by specifying its constituent processes. (Fig. 1)

There are three processes: AVERAGE, PLOT and COMMAND.

The averaging process has to work in real time, while the plotter is rather slow, so that there is a great advantage in having them independent and asynchronous. The structure of these processes is simple, they are either working or idling.

Process COMMAND is driven by the operator using a teletype and is designed so that it has four states. From the idling condition (state 0) the operator may start the averaging procedure, or start the plotting procedure, (assuming that there is something to plot). While plotting he may desire to start another average or alternatively while averaging to start a plot. There are therefore three further states, averaging (state 1) during which he cannot start another average, and plotting (state 2) during which he cannot start another plot, and finally averaging and plotting (state 3) during which he can do nothing but wait.

This system has theoretically 256 possible compound states (2 for process average, 2 for plot, 4 for command and 224 for the possible states of the four routes—each route may either contain a message or be empty). It can in fact only assume 16 states and the validity is easily checked.

If however the simple mistake is made of letting process COMMAND return from state 3 to state 0 whenever it receives a message that the plotting or averaging has finished, then 224 of the possible states are reached and there are six groups of states in which no progress can be made.

It is therefore no surprise that as yet we have been unable to analyse fully the complete working program, which has several additional processes, and it seems that more powerful methods need to be developed before such analyses can become really useful for practical problems.

Yours faithfully,

Dr. Graham Hill
Dr. H. R. A. Townsend

Regional Clinical Neurophysiology Service
Western General Hospital
Crewe Road
Edinburgh
EH4 2XU

25 July 1974

```
PROCESS (AVERAGE)
GET 0 0->1         (GETS MESSAGE TO START EXPERIMENT)
PUT 1 1->0         (REPLIES AFTER A PRESET TIME WITH RESULT)
;

PROCESS (PLOT)
GET 2 0->1         (GETS MESSAGE CONTAINING DATA FOR PLOTTING)
PUT 3 1->0         (REPLIES WHEN FINISHED)
;

PROCESS (COMMAND)
PUT 0 0->1         (START AVERAGING PROCEDURE)
GET 1 1->0         (RECEIVE RESULTS)
PUT 0 2->3         (START NEW EXPERIMENT WHILE PLOTTING)

PUT 2 0->2         (START PLOTTING)
GET 3 2->0         (RECIEVE MESSAGE, PLOT COMPLETE)
PUT 2 1->3         (START PLOTTING WHILE EXPERIMENT IN PROGRESS)

GET 1 3->2
GET 3 3->1

;
```

Fig. 1

```
     PROCESSES        ROUTES

RECURRENT COMPLETE TERMINAL CLASS    0

   0:  0  0  0      0 0 0 0
   1:  0  0  1      1 0 0 0
   2:  0  0  2      0 0 1 0
   3:  1  0  1      0 0 0 0
   4:  0  0  3      1 0 1 0
   5:  0  1  2      0 0 0 0
   6:  0  0  1      0 1 0 0
   7:  1  0  3      0 0 1 0
   8:  0  1  3      1 0 0 0
   9:  0  0  2      0 0 0 1
  10:  0  0  3      0 1 1 0
  11:  1  1  3      0 0 0 0
  12:  0  0  3      1 0 0 1
  13:  0  1  3      0 1 0 0
  14:  1  0  3      0 0 0 1
  15:  0  0  3      0 1 0 1
```

Fig. 2

```
     PROCESSES        ROUTES

RECURRENT TERMINAL CLASS   49

137:  0  1  0       0 0 1 1
148:  0  1  1       1 0 1 1
160:  1  1  1       0 0 1 1
172:  0  1  1       0 1 1 1
WARNING: NO CHANGE IN ROUTE    2
WARNING: NO CHANGE IN ROUTE    3

RECURRENT TERMINAL CLASS   51

147:  1  0  0       1 1 0 0
159:  1  0  2       1 1 1 0
171:  1  1  2       1 1 0 0
181:  1  0  2       1 1 0 1
WARNING: NO CHANGE IN ROUTE    0
WARNING: NO CHANGE IN ROUTE    1

RECURRENT TERMINAL CLASS   58

188:  0  1  0       1 0 1 1
193:  1  1  0       0 0 1 1
198:  0  1  0       0 1 1 1
199:  1  1  1       1 0 1 1
206:  0  1  1       1 1 1 1
211:  1  1  1       0 1 1 1
WARNING: NO CHANGE IN ROUTE    2
WARNING: NO CHANGE IN ROUTE    3

RECURRENT TERMINAL CLASS   60

192:  1  0  0       1 1 1 0
197:  1  1  0       1 1 0 0
204:  1  0  0       1 1 0 1
205:  1  1  2       1 1 1 0
210:  1  0  2       1 1 1 1
214:  1  1  2       1 1 0 1
WARNING: NO CHANGE IN ROUTE    0
WARNING: NO CHANGE IN ROUTE    1

RECURRENT TERMINAL CLASS   62

216:  1  1  0       1 0 1 1
218:  0  1  0       1 1 1 1
220:  1  1  0       0 1 1 1
222:  1  1  1       1 1 1 1
WARNING: NO CHANGE IN ROUTE    2
WARNING: NO CHANGE IN ROUTE    3

RECURRENT TERMINAL CLASS   63

217:  1  1  0       1 1 1 0
219:  1  0  0       1 1 1 1
221:  1  1  0       1 1 0 1
223:  1  1  2       1 1 1 1
WARNING: NO CHANGE IN ROUTE    0
WARNING: NO CHANGE IN ROUTD    1

ERROR:    6 TERMINAL CLASSES
```

Fig. 3

Erratum

The following references cited by J. K. Broadbent in his paper *Microprogramming and system architecture* published in *The Computer Journal* (February 1974) should not be attributed to Rosin *et al.* but are in fact computer manufacturers' publications:

(*a*) Meta 4 Series 16 Computer System, Preliminary System Manual, San Diego, March 1970.

(*b*) Interdata Model 85, Dynamic Control Store application Guide, New Jersey, 1972.

(*c*) Microprogramming, Varian, 1972.

The author and the editor regret any embarrassment or confusion which may have been caused.