more strongly the need for systems analysts to be taught more about the ways in which computers can serve commerce and industry, and not only about *management aspects* and *the users' point of view* (although he may have implied this). Furthermore, he nowhere shows the need for a thorough induction period for all new entrants, and perhaps for periodical cross-posting with various user departments.

Will Mr. Coates please accept my apologies for these curmudgeon-like remarks: his article remains undiminished by them.

Yours faithfully,

J. C. VORVOREANU

Senior Consultant
Data Logic Limited
Westway House
320 Ruislip Road East
Greenford
Middlesex UB6 9BH
24 September 1974

To the Editor
The Computer Journal

Sir

A proof of the radix conversion process described by Boothroyd in *The Computer Journal* (Vol. 17, No. 1, p. 95) can be derived as follows: Let

$$(X)_r = (?)_b$$

$$(X)_r = \sum_{i=0}^{n} a_i \cdot r^i \qquad (1)$$

$$= a_n r^n + a_{n-1} r^{n-1} + a_{n-2} r^{n-2} + \ldots + a_1 r^1 + a_0 r^0$$

It remains to be shown that the final result is equivalent to (1).

The number $(X)_r$ is to be manipulated in the radix b, that is it is treated as a number X^1 where

$$X^1 = a_n b^n + a_{n-1} b^{n-1} + a_{n-2} b^{n-2} + \ldots + a_1 b^1 + a_0 b^0 \ .$$

The appropriate multiplier is $(b - r)$.

Consider the first term in X^1.

(i) Multiply by $(b - r)$

$$= a_n b^{n+1} - a_n r^1 b^n \ .$$

(ii) Shift right 1 place (i.e. divide by b)

$$= a_n b^n - a_n r^1 b^{n-1} \ .$$

(iii) Subtract from X^1

$$\begin{array}{l} a_n b^n + a_{n-1} b^{n-1} + a_{n-2} b^{n-2} + \ldots a_1 b^1 + a_0 b^0 \\ - \underline{a_n b^n - a_n r^1 b^{n-1}} \\ (a_{n-1} + a_n r^1) b^{n-1} + a_{n-2} b^{n-2} + \ldots + a_1 b^1 + a_0 b^0 . \end{array}$$

Repeating operations (i)-(iii), the result will be

$$(a_{n-2} + a_{n-1} r^1 + a_{n-2} r^2) b^{n-2} + \ldots\ldots + a_1 b^1 + a_0 b^0 \ .$$

After n manipulations, the result will be

$$b^{n-n}(a_{n-n} + a_{n-n+1} r^1 + a_{n-n+2} r^2 + \ldots + a_{n-1} r^{n-1} + a_n r^n)$$

$$= b^0(a_{n-n} + a_{n-n+1} r^1 + a_{n-n+2} r^2 + \ldots\ldots + a_{n-1} r^{n-1} + a_n r^n$$

$$= a_0 + a_1 r^1 + a_2 r^2 + \ldots\ldots + a_{n-1} r^{n-1} + a_n r^n$$

$$= \sum_{i=0}^{n} a_i r^i \text{ Q.E.D.}$$

Yours faithfully,

J. TOMLINSON

St. Albans Training and Education Centre
Post Office Data Processing Service
25 Grosvenor Road
St. Albans
Hertfordshire
5 April 1974

To the Editor
The Computer Journal

Sir

The radix conversion process described by Mr. J. Boothroyd in his letter in *The Computer Journal* (Vol. 17, No. 1) can be proved valid for integers as follows.

Let P be integer and $(P)_b$ be its representation with radix b then

$$(P)_b = b^n x_n + b^{n-1} x_{n-1} + \ldots + b^1 x_1 + b^0 x_0 \qquad (A)$$

where x_is are digits with radix b.

The above definition (A) can be written in an algorithmic way as follows.

ALGORITHM B

1. $m_n = x_n$
2. $m_i = b \cdot m_{i+1} + x_i$
   for $i = n - 1, n - 2, \ldots\ldots, 2, 1, 0.$

where x_is and b are represented with radix b and arithmetic in step 2 is performed with radix b.

Then m_0 yields P with radix b i.e. $m_0 = (P)_b$.

For conversion of radix, say from b to e what is needed is to change:

(*a*) a representation of b and x_is from radix b to radix e,

(*b*) and arithmetic in step 2 of ALGORITHM: B is performed with radix e.

Then m_0 yield $(P)_e$.

The ALGORITHM B can be written with a slight modification in step 2 as follows.

ALGORITHM C

1. $m_n = x_n$
2. $m_i = e \cdot m_{i+1} + x_i + (b - e)\, m_{i+1}$
   for $i = n - 1, n - 2, \ldots\ldots\ldots, 2, 1, 0.$

Now if e, b and x_is are represented with radix e and arithmetic is performed with radix e then $m_0 = (P)_e$.

ALGORITHM C is the conversion process described by Mr. J. Boothroyd. Note that for conversion of binary to decimal ALGORITHM B is faster, while for conversion of octal to decimal ALGORITHM C is faster.

Yours faithfully,

S. N. BALDOTA

E.D.P. Centre
University of Bombay
Bombay 400 020
India
1 August 1974

To the Editor
The Computer Journal

Sir

There is an error in the argument of the paper by Kohavi, Rivierre, and Kohavi (1972) for the derivation of formulae for high-order identifying sequences in reduced, strongly-connected automata. It is assumed that each characterising sequence will assign the state to be identified s_j, to a k-state block. In general, this does not hold: each characterising sequence x_i will assign s_j to a block containing k_i states ($k_i <$ number of states in the machine). Thus, when $Y_i = X_i T(R_i, S_j)$ is applied with the correct response in the identifying sequence I_j, the state to which it was applied is assigned to a k_i-state block B_i containing S_j. So, we need to apply $Y_i k_i + 1$ times to ensure that we are still at a state in B_i before applying Y_{i+1}. The formula for third and fourth order identifying sequences arrived at in this way are respectively:

$$I_j = (Y_1{}^{k_1+1} Y_2)^{k_2+1} Y_1{}^{k_1+1} Y_3 \qquad (1)$$

$$I_j = ((Y_1{}^{k_1+1} Y_2)^{k_2+1} Y_1{}^{k_1+1} Y_3)^{k_3+1} (Y_1{}^{k_1+1} Y_2)^{k_2+1} Y_1{}^{k_1+1} Y_4 \qquad (2)$$

Both (1) and (2) can be derived from the following reduction formula for nth order identifying sequences:

$$\begin{array}{l} Z_1 = Y_1 \\ {}^*Z_{i+1} = Z_i{}^{k_i+1} Z_i\, [Y_i / Y_{i+1}] \\ I_j = Z_n \ . \end{array} \qquad (3)$$

This latter formula can be improved on if we note that before a correct response to Y_i, not only does the state being tested belong to block B_i, but to $\hat{\imath}\, B_i$. Therefore, only $h_i = |\hat{\imath}\, B_i|$ states exist that can elicit the correct response to the sequence so far. This means that Y_i and its prefix need only be applied with correct response $h_i + 1$ times to ensure that Y_{i+1} is being applied to a state known to belong to $\hat{\imath}\, B_i$. So we can change reduction formula (3) to:

$$\begin{array}{l} Z_1 = Y_1 \\ h_i = |\hat{\imath}\, B_i| \\ {}^*Z_{i+1} = Z_i h_i{}^{+1} Z_i\, [Y_i / Y_{i+1}] \\ I_j = Z_n \ . \end{array} \qquad (4)$$

The effect of this latter improvement is to increasingly reduce the length of the sequence derived from (4) in comparison to that derived from (3) with increasing order of the sequence, thereby providing large savings in testing time when the order is not small i.e. 1 or 2.