A simple algebra system

By D. Barton,[*] S. R. Bourne[†] and C. J. Burgess[‡]

This paper describes a computing system that enables a particular class of algebraic manipulative problems to be simply programmed. The scheme is described by reference to an example of a problem involving the derivation of a function that may potentially contain many thousands of terms.

(Received February 1968)

The system that we wish to describe here may be conveniently divided into two sections. The first section is concerned with the language in which the manipulative problem is to be programmed. This language will be described by an example. We shall not be concerned with the compiling techniques since these have been fully described elsewhere (Matthewman, 1966). The second section contains the run-time system and is composed of a set of closed subroutines that actually carry out the manipulation under the direction of the compiled object code. This latter system of programs has been described in Barton (1967) but a brief specification of the facilities available is desirable here in order to draw attention to the limitations of the system and to indicate possible means of extension. It is important to remark that while the syntax of the language that we shall describe is almost identical to Titan Autocode (—, 1967) the data elements with which we compute are not numbers but algebraic expressions.

The run-time system

This package of programs is able to perform elementary algebraic operations on functions of the form

$$\alpha = \sum_{\substack{i\,j\,k \\ i'j'k'}} \mathrm{P}^{ijk}_{i'j'k'}(a, b, \ldots, l) \begin{matrix}\cos\\ \sin\end{matrix} (iu \pm jv \pm \ldots \pm k'z) \quad (1)$$

where $\mathrm{P}^{ijk}_{i'j'k'}$ is a polynomial in the variables $a, b, \ldots, l$ with coefficients in the rational field. The numbers $i, j, \ldots, k'$ are positive integers and the summation is finite. The run-time system provides subroutines for the operations of addition, multiplication, differentiation, integration and substitution for the functions α defined in equation (1).

The principal limitations of the system are imposed by the restriction to a particularly simple class of functions such as those in equation (1). In the course of our work two of us are normally engaged in algebraic calculations derived from celestial mechanics and we have found the scheme entirely adequate for our needs. Experience with problems in engineering and theoretical physics, however, indicates that the run-time system should be upgraded to compute with polynomials in a greater number of variables. Perhaps surprisingly in the problems that we have considered there has been no requirement for a simplification algorithm or for more complex functions. We have, however, found it necessary to allow for polynomials with double length rational coefficients, complex coefficients and floating point coefficients, the range of single length integers being $(0, 10^{11})$. These modifications are incorporated in a new run-time system that allows an arbitrary number of polynomial variables and is in the process of development.

The programming language

The language in which programs are written allows the four types of variable listed below:

1. *Indices*

These are fixed point signed integers in the range $|n| \leqslant 10^6$ and are called by the names $I, J, \ldots, S, T$. It is also possible to use arrays of indices indexed by a single modifier. These are called as $I[n], \ldots, T[n]$.

2. *Expressions*

These are literal functions of the type α (equation 1). They may be called by any of the names $A, B, \ldots, H$ or $U, V, \ldots, Z$. Arrays of such expressions are allowed and these are called $A[n], \ldots, H[n], U[n], \ldots, Z[n]$.

3. *Polynomial variables*

These are the atomic variables $a, b, \ldots, l$ that occur as arguments in the polynomials that make up an expression defined in equation (1).

4. *Periodic variables*

These are the atomic variables $u, v, \ldots, z$ that occur as arguments of cosine or sine in an expression α.

The arithmetic operations carried out on expressions are denoted by $+$ and $-$, while multiplication is denoted by juxtaposition. However, we have used . to represent exponentiation, * for differentiation and $ for integration of expressions. The differential coefficient of an expression whose name is A with respect to atomic variable a is therefore written $A{*}a$, while the integral is written $\$Aa$. Thus the sequence of instructions $A = a.2$; $A = A + A{*}a$; $A = \$(Ab)b$ results in the setting of A to the expression $\frac{1}{2}a^2b^2 + ab^2$.

* *University College, Cambridge.* † *Trinity College, Cambridge.* ‡ *Computer Unit, The University, Bristol.*

Since there is no explicit multiplication operator we have used quotes to allow the input of complex expressions including cosine and sine. Hence $A = 'SIN(U)'$ sets A to $\sin(u)$. The symbol | (vertical bar) has been used to introduce comment in programs and everything following on the same line as | is ignored, including |.

Two substitution routines are provided by the system and these will be illustrated by example. The program $A = a + b$; $B = c.2$; $B =$ SUBSTITUTE (B, A, c) will result in the setting of variable B to the expression $a^2 + 2ab + b^2$, where we have substituted the initial value of A into the initial value of B for the polynomial variable c. The second type of substitution that is provided will perform as follows:

$$A = 'COS(U)';$$

$$B = a;\ C = \text{SUBSTITUTE } (A, v + B, u, 4)$$

will result in C being set to the expression

$$\cos(v)\left(1 - \frac{a^2}{2!} + \frac{a^4}{4!}\right) - \sin(v)\left(\frac{a}{1!} - \frac{a^3}{3!}\right).$$

Here one should note that the literal expression $\cos(u)$ is enclosed in quotes. The substitution program has then substituted into the initial value of A for the periodic variable u the expression $v + B$. The expression B is assumed to be a small quantity and the result is the appropriate Taylor series in the initial value of B truncated to include only the terms up to order 4, the last parameter in the call of substitute.

In the above example the algebraic operations that have been performed by the call of SUBSTITUTE are

(*a*) u is replaced by $v + B$ in the expression $\cos(u)$.

(*b*) The expansion

$$\begin{aligned}\cos(v + B) &= \cos v \cos B - \sin v \sin B \\ &= \cos v\left(1 - \frac{B^2}{2!} + \frac{B^4}{4!} \dots\right) \\ &\quad - \sin v\left(\frac{B}{1!} - \frac{B^3}{3!} + \dots\right)\end{aligned}$$

is performed.

(*c*) The expression B is replaced by its value a, to obtain the result.

Our work in celestial mechanics frequently requires the expansion of functions as Taylor series in small parameters and the truncation of such series to include terms less than some prescribed order. The run-time system therefore provides a facility to check the result of any calculation for the presence of unwanted terms. These checks are imposed by the run-time system whenever it carries out a multiplication and may be used in several modes. Mode zero is preset and in this mode all multiplication is carried out exactly. Mode one will select terms whose total order is less than some prescribed quantity, while mode two will select only those terms whose order in a particular variable is less than a given quantity. Several other modes are available. The particular mode is selected by the directive MULTIPLY that takes effect at run-time. The directive defines what is meant by multiplication and takes two parameters. The first is simply an integer indicating which particular mode of multiplication is to be employed, and the second parameter references an index vector containing the parameters associated with each mode of multiplication. For example, $A = ab$; $B = a + b.2$; MULTIPLY$(1, I)$; $I[0] = 3$; $A = AB$ will set the variable A to the expression a^2b since multiplication in mode one selects only those terms whose total order is less than or equal to the first parameter of the index array, namely $I[0]$.

An example of the use of the system

The calculation that we shall present as an example is drawn from celestial mechanics. It is a calculation that potentially involves an enormous quantity of routine algebra but it is otherwise a very simple problem. We require to produce an expansion of the function F given by

$$F = X_1^2 Y_2^3 (P_2(X) + (\gamma) X_1 Y_2 P_3(X) + (\gamma)^2 X_1^2 Y_2^2 P_4(X) + \dots) \quad (2)$$

to a given order in the small quantities a, b and c, where the following relations apply:

$$\left.\begin{aligned} u &= E - a\sin E & v &= E' - b\sin E' \\ X_1 &= 1 - a\cos E & Y_1 &= 1 - b\cos E' \\ X_2 &= 1 + \frac{dE}{du} & Y_2 &= 1 + \frac{dE'}{dv} \end{aligned}\right\} \quad (3)$$

together with

$$\left.\begin{aligned} X &= (1 - e^2)\cos(\phi - \phi') + e^2\cos(\phi + \phi'), \\ \phi &= u + w + \int \surd(1 - a^2) X_2^2 du \\ \text{and } \phi' &= v + x + y - z + \int \surd(1 - b^2) Y_2^2 dv. \end{aligned}\right\} \quad (4)$$

The $P_n(X)$ are the Legendre Polynomials. The final expansion of F is to be in terms of the variables a, b, e, u, v, w, x, y and z together with γ. Since γ is to be treated as a quantity of degree two it will be represented by c^2 where c is a literal variable.

We proceed to solve the first of equations (3) in the form $E = E(u, a)$ and hence to obtain X_1 and X_2 in terms of u and a. The change of variables $a \to b$, $u \to v$ leads to the expressions for Y_1 and Y_2 in terms of v and b. We may then evaluate ϕ, ϕ' and hence X from the first of equations (4) and finally substitute these values into (2) to obtain F.

To solve the equation $u = E - a\sin E$ we write $E = u + \mathscr{A}$ and notice that the zero order approximation to $\mathscr{A}$ is $\mathscr{A}_0 = 0$. Given that the nth order approximation to $\mathscr{A}$ is $\mathscr{A}_n$ it follows at once that the next approximation is given by

$$\mathscr{A}_{n+1} = a\sin\mathscr{A}_n. \quad (5)$$

We may therefore write the following simple program to calculate $E = E(u, a)$.

```
$X[0] = 0$; MULTIPLY $(1, I)$              | Set initial approximation
                                           | and arrange to include
                                           | particular terms.
FOR $I = 1{:}1{:}N$; $I[0] = I$            | Select terms degree $I$
$X[0]$ = SUBSTITUTE ('$A$ SIN($U$)', $u + X[0]$, $v$, $I$)
REPEAT
```

The result of this calculation is an expression for $E(u, a)$ stored in $X[0]$ and accurate to order n in a. In the above program great power is provided by the use of a FOR-REPEAT loop using the index variable I. A long and tedious algebraic procedure is thus performed by a program of extreme simplicity.

It will be seen in equation (4) that we require a power series approximation to $\sqrt{(1 - a^2)}$ and while it would be possible to input such a series directly it is in practice more convenient to derive it as a solution of the equation $C^2 = 1 - a^2$. Once again by an approximation technique where C_n is the nth order approximation to C we have

$$C_{n+1} = (1 - a^2 - C_n^2)/2 + C_n \qquad (6)$$

and since the first approximation to C is $C_0 = 1$ we may calculate C using the following program:

```
$C = 1$; $D = (1 - hh - aa)/2$             | Set initial approximation
MULTIPLY $(1, I)$                          | and the value of (6).
FOR $I = 1{:}1{:}N$; $I[0] = I$
$C = C +$ SUBSTITUTE $(D, C, h)$
REPEAT
```

Finally our problem requires us to substitute into the Legendre polynomials $P_n(X)$ for X and we shall therefore require an explicit representation of these polynomials. They may be calculated and assigned to the vector $A[1] \ldots A[N]$ by the program

```
PROGRAM θ A[4] X[3] Y[3] I[θ] |PROGRAM TO CALCULATE THE DISTURBING FUNCTION

1:
|fix the order of derivation
N=4

|calculate the first few Legendre polys
FOR I=1:1:N
A[I]=(hh-1).I ; FOR J=1:1:I ; A[I]=(A[I]*h)/(2J) ; REPEAT
REPEAT

|arrange to round all products down to order less than or equal to I[θ]
I[θ]=N ; MULTIPLY(1,I)

|calculate square-root of (1-aa) and E=E(u,a)
D=(1-hh-aa)/2 ; C=1 ; X[θ]=θ
FOR I=1:1:N
I[θ]=I ; C=C+SUBSTITUTE(D,C,h) ; X[θ]=SUBSTITUTE(a'SIN(V)',u+X[θ],v,I)
REPEAT

|calculate X1 X2 X3
X[1]=SUBSTITUTE(1-a'COS(V)',u+X[θ],v,N) ; X[2]=1+X[θ]*u ; X[3]=$(X[2]X[2]C)u

|change variables a to b u to v to obtain Y2 Y3
FOR I=2:1:3 ; Y[I]=SUBSTITUTE(SUBSTITUTE(X[I],v,u,1),b,a) ; REPEAT

|calculate X
X=(1-ee)'COS(U-V)'+ee'COS(U+V)'
X=SUBSTITUTE(SUBSTITUTE(X,u+w+X[3],u,N),v+x+y-z+Y[3],v,N)

|calculate  X1 Y2 and X1^2 Y2^3
Z=X[1]Y[2] ; Y=ZZY[2]

|substitute to obtain Pn(X) for n=2,3,...
FOR I=2:1:N ; I[θ]=N+4-2I ; A[I]=SUBSTITUTE(A[I],X,h) ; REPEAT ; I[θ]=N

|calculate and print the disturbing function
FOR I=N:-1:3 ; A[N]=ccZA[N]+A[I-1] ; REPEAT ; PRINT(YA[N])

|directives to stop program and to finish compiling and run program
STOP ; START 1
```

Fig. 1

```
FOR $I = 1:1:N$
  $A[I] = (hh - 1).I$
  FOR $J = 1:1:I$; $A[I] = A[I]*h/2J$; REPEAT
REPEAT
```

Here we have used Rodrigues' formula

$$P_n(X) = \frac{1}{2^n n!}\frac{d^n}{dh^n}(h^2 - 1)^n.$$

The remainder of the problem is composed simply of elementary substitutions and requires no explanations. A complete program is presented in **Fig. 1** together with the set of results obtained for $N = 2$.

It should be understood that the calculation described above when taken to the 4th order results in an expression containing many hundreds of terms, while an 8th order development contains many thousands of terms. Nevertheless the simplicity of the actual calculation is preserved in the program and the cost in terms of computing power (time and store used) is remarkably small.

The complete program is shown in Fig. 1 and it is introduced by the words PROGRAM 0. This is followed by a specification of the arrays required during the run. The program begins at label 1 as directed by the instruction START 1 at the end of the program.

The output for this program is presented in **Fig. 2** and is an exact reproduction of the printed output. The independent periodic terms in the expression are numbered 1–21 by the numbers in the left-hand margin. The numbers in the right-hand margin refer to the number of polynomial terms in the coefficient of the corresponding periodic term. The capital letters that occur in Fig. 2 correspond to the lower case letters $a, b, \ldots l$ that are atomic variables to the compiler. Periodic term 16 is arranged over three lines and is in fact

$$\left(\frac{3}{4} - \frac{3}{2}e^2 - \frac{15}{8}b^2 - \frac{15}{8}a^2\right)\cos(2u - 2v + 2w - 2x - 2y + 2z).$$

The actual result should be interpreted as the sum of the 21 periodic terms.

A compiler for the system

A load and go compiler has been written for the language using Psyco (Matthewman, 1966) and at compile-time this occupies 16K of core store. At run-time the store required depends upon the size of the calculation involved. For the above program 8K is required when $N = 2$, while 12K is necessary for the case $N = 4$. (The Titan is a prototype Atlas 2 with 128K of 48-bit words.) The time required to compile and run the case $N = 2$ is 35 seconds, and for $N = 4$ is 90 seconds.

The system we have described is particularly well suited to the calculation that we have used as an example, and it is in daily use performing the calculations for which it was designed. Indeed the existence of this system has made it possible to undertake many calculations that would otherwise be impracticable in a reasonable time. It is clear that this system is restricted in its scope, and it is through the manipulative routines that most of the system's restrictions arise. It should be mentioned that the compiler is quite independent of these routines. However, there is evidence that the modifications indicated earlier in this paper will improve the run-time system to such an extent that it will be of real value to users other than those in celestial mechanics.

```
 1)      <+ 1/4  - 3/2E.2 + 3/8B.2 + 3/8A.2>                    (4
 2)       + 3/4B COS(V)                                          (1
 3)       + 9/8B.2 COS(2V)                                       (1
 4)       + 3/2E.2 COS(2V+2X+2Y-2Z)                              (1
 5)       + 15/8A.2 COS(2V-2W+2X+2Y-2Z)                          (1
 6)       - 1/2A COS(U)                                          (1
 7)       - 3/4AB COS(U+V)                                       (1
 8)       - 63/8AB COS(U-3V+2W-2X-2Y+2Z)                         (1
 9)       - 9/4A COS(U-2V+2W-2X-2Y+2Z)                           (1
10)       - 3/4AB COS(U-V)                                       (1
11)       + 9/8AB COS(U-V+2W-2X-2Y+2Z)                           (1
12)       - 1/8A.2 COS(2U)                                       (1
13)       + 3/2E.2 COS(2U+2W)                                    (1
14)       + 51/8B.2 COS(2U-4V+2W-2X-2Y+2Z)                       (1
15)       + 21/8B COS(2U-3V+2W-2X-2Y+2Z)                         (1
16)                      COS(2U-2V+2W-2X-2Y+2Z)
                         ----------------------
         <+ 3/4  - 3/2E.2 - 15/8B.2 - 15/8A.2>                   (4
17)       - 3/8B COS(2U-V+2W-2X-2Y+2Z)                           (1
18)       + 21/8AB COS(3U-3V+2W-2X-2Y+2Z)                        (1
19)       + 3/4A COS(3U-2V+2W-2X-2Y+2Z)                          (1
20)       - 3/8AB COS(3U-V+2W-2X-2Y+2Z)                          (1
21)       + 3/4A.2 COS(4U-2V+2W-2X-2Y+2Z)                        (1
```

Fig. 2

Some further remarks on the complete algebra system

The above examples have indicated how the compiler should be used to perform simple algebra. However, the bare bones of a system such as ours would be of little value unless it contained facilities for program control, together with service routines allowing complete control of space inside the computer, and comprehensive input/output facilities. Further, such a system must contain both a compile-time and run-time diagnostic routine.

Program control is similar to that provided by Titan Autocode and consists of conditional and unconditional jumps together with 'For-Repeat' loops.

Space at run-time is handled at the lowest level by the run-time system described in Barton (1967). It is a feature of that system that whenever it is possible to perform a calculation involving one or two operands and derive the result entirely within the space occupied by the operands, then this is done and the operands are deleted. However, it is not desirable that the compiler instruction $A = B + C$ should result in the deletion of expressions B and C whenever it is obeyed. Consequently the compiler must copy B and C and add the copies before assigning to A. This implies that if B and C are both of such size that they occupy more than half of the available space then the instruction $A = B + C$ cannot be successfully obeyed. A facility is therefore provided that ensures that a labelled list name is not unnecessarily copied when an arithmetic expression is evaluated, and the instruction $A = B{:} + C{:}$ adds B to C without copying either and hence deleting both. In our work we have found this facility to be extremely useful both as a time and a space economy. In some circumstances when it is not convenient to use this labelling facility the space occupied by the expression A may be returned to the pool of free space by the instruction LOSE(A).

Data may be input to the system in several ways. First, explicit expressions may be written into a program, e.g. $A = a$ or $A =$ 'SIN(U)'. Alternatively, an expression may be input by the instruction $A =$ INPUT that causes the next complete expression from the currently selected input stream to be read in and assigned to A. The stream is administered entirely by the Titan supervisor, and information is presented to the compiler from the stream one character at a time. Other streams may be selected by a READER instruction, and there may be up to 15 co-existent input streams. The expressions presented on an input stream may be written either in free format and delimited by a closing bracket followed by a new line, or in fixed format as output by the compiler.

Output from the compiler may be obtained on up to 15 separate output streams selected by the OUTPUT instruction, and the instruction PRINT provides for output of both expressions and indices. Text may be output by the TEXT instruction.

The fixed format output from the compiler, while being comparatively simple to read, contains considerable redundancy and it is therefore convenient to provide binary input/output for the storage of expressions that are required for further calculation. For this purpose the READ, WRITE and TAPE instructions are provided and these arrange for a binary copy of an expression to be written or read from a specified block of a particular magnetic tape.

In order to avoid recompiling a working program, facilities have been provided to write a complete restartable core image of a program and its data area to magnetic tape at run-time. This core image may later be reloaded and restarted either at a prescribed point after re-initialisation or alternatively immediately after the dumping point. By using this facility the user can arrange to run his program in the minimum of core store since, if at any stage the store allocation requires to be changed, the program may simply be dumped and restarted with a new allocation.

At compile-time syntax errors are discovered by the compiler and the offending line is printed out. Compilation continues to the end of the program and generally all syntax errors are indicated by one test run with the compiler. At run-time any error that occurs leads to a diagnostic listing that contains the line number of the offending instruction, a message describing the fault, the values of all indices, and a list of those variables that have been used and those that are still in use. Any attempt to compute with a variable that has not been set, or one whose space has been relinquished and therefore has no value, leads to an error.

Any run-time fault may be trapped using a TRAP instruction, and when the fault occurs control will be transferred to a routine in the user's program so that he may arrange his own diagnostics or take some other action following the error.

A facility is provided in the present system to allow work involving the symbol $\sqrt{-1}$. The polynomial variable i is treated as $\sqrt{-1}$ in all circumstances. It should be noted that it is necessary to provide a complex number facility in order to allow work with complex double length rational numbers that are required for some applications in celestial mechanics.

Conclusion

We have indicated above that this compiler is independent of the run-time system as far as the manipulative routines are concerned. It would seem that a useful purpose would be served by the development of a simple language, the syntax of which was defined, while the semantics were left undefined, at least in so far as arithmetic operations are concerned. It would then be possible for independent users to construct their own systems for algebraic manipulation or indeed other projects. These systems could be specially tailored to meet the requirements of particular problems and would, in general, be more economic in time and store than the comprehensive giant systems that will otherwise be developed. Further, it is remarkable that in no problem that we have encountered has it been necessary to factorise polynomials or general expressions or even to simplify expressions in anything other than the trivial sense.

We are of course aware that simplification and factorisation are of great importance in certain problems of mathematics that it is desirable to solve on a computer, but nevertheless it is unnecessary to hold up possible developments on other subjects while these very difficult problems are overcome.

It is true that languages have been previously developed with a syntactic/semantic distinction similar to that indicated above but these do not appear to be generally available to the machine user outside the computing laboratory.

We should like to thank Dr. J. H. Matthewman for the use of the Psyco compiler with which we have constructed our own compiler. Further, we thank him for his invaluable assistance in teaching us to use Psyco and assisting with the development of the compiler. For extensive use of the computing facilities available in Cambridge it is our pleasure to thank the Director and Staff of the Mathematical Laboratory.

References

MATTHEWMAN, J. H. (1966). Ph.D. Thesis, Cambridge University.
BARTON, D. (1967). A scheme for manipulative algebra on a computer, *Computer Journal*, Vol. 9, p. 340.
——, (1967). *Titan Autocode programming manual*, University Mathematical Laboratory, Cambridge.

Book Review

New Methods of Thought and Procedure, edited by F. ZWIKCY and A. G. WILSON, 1967; 338 pages. (*Springer*, 79s.)

This book contains contributions to a Symposium in Methodologies sponsored by the Office for Industrial Associates of the California Institute of Technology and the Society for Morphological Research, held at Pasadena in May 1967. The contents include a short prologue by Zwicky, six sections on Operations Research, Systems Engineering, Dynamic Programming, Information Theory, Game Theory and Morphological Research ending with an epilogue by Wilson.

The opening paragraph of the epilogue perhaps best sums up the aspirations of the organisers. Wilson says 'A primary purpose of this conference has been to consider whether the various methodologies employed in solving problems when taken together constitute in themselves a useful scientific and technological discipline. The descriptions of the several approaches to problems that have been presented here—Operations Research, Systems Engineering, Morphological Analysis, etc.—have made visible some common principles which have been independently developed for structuring, analysing, and solving complex problems of many types. Though using different names and terminologies, the identities and overlaps contained in these approaches, taken with the fact of their independent discovery in many diverse contexts, strongly suggest the developability of a useful discipline that we may call 'methodology'. Although the presentations during this conference have only partially defined the subject area of methodology, thay have demonstrated that it would now be meaningful to take steps towards systematic definition and organisation of the concepts so far developed and establish a formal discipline.' Zwicky in the chapter of Section VI on Morphological Research defines his approach thus: 'The morphological approach to discovery, invention, research and construction has been conceived and developed for the purpose of dealing with *all* situations in life more reasonably and more effectively than hitherto. This is achieved through the study of all relevant interrelations among objects, phenomena and concepts by means of methods which are based on the utmost detachment from prejudice and carefully refrain from all prevaluations. Applications of the morphological methods of the total field coverage, of negation and construction, of the morphological box and others to technical problems in particular and to human problems in general are described. These not only illustrate how discovery, invention and research can be conducted most effectively but also how the morphological approach makes possible the clear recognition of those fatal aberrations of the human mind which must be overcome if we are ever to build a sound world.'

So far as the individual contributions are concerned, I doubt if the Conference was really a success from the standpoint of the editors. Each section contains at least one contribution of real merit—Bellman's chapter 'Dynamic Programming: A Reluctant Theory' is, to me at any rate, a wholly convincing and delightful description of how he got into Dynamic Programming, but like most contributions appears to add little to realising the aims of the Symposium, unless it is the claim that almost every problem involving decision-making is a problem involving multi-stage decision-making to the solution of which dynamic programming provides a comprehensive approach.

Most Operational Research workers would probably agree that problems need to be approached from many angles and can seldom be isolated completely from their context. Not enough people trying to instal computers in industry and commerce realize that the accounting systems, techniques of stock control and other management sciences which they seek to computerize are in fact devices to control a working system; the implementation processes involve the destruction of the former control system and the creation of a new one which when embedded in the working system interacts with it and more often than not causes it to change. This failure to appreciate the 'ecological' balance between various parts of a system explains, in my view, many of the failures of industrial and commercial computer installations. Thus, it must be concluded that Zwicky has a case to argue. In his own chapter, however, I believe he goes too far, at least in so far as his description of devising a missile launching system is concerned. For he proposes that the morphologist will study the whole class of possible solutions to the problem, before reaching a decision. There is a danger in this that no progress at all will be made since the results of all possible research—future as well as past—have to be considered.

Having suggested that the contents of the book do not, as a whole, achieve the editors' purposes, I can nevertheless recommend it to readers. Most of the individual contributions have a value in their own right and are well worth study. No one could fail to profit by reading the excellent papers on Systems Engineering by Dean Gillette of Bell Telephone and on Games Theory by Oscar Morgernstern.

A. YOUNG (Liverpool)