An extension of binary minimisation techniques to ternary equations

By S. L. Hurst*

This paper describes an extension of well-known binary minimisation techniques to cover the case where the given variables have three possible states, termed 0, 1, and 2. Both graphical and numerical minimisation techniques are considered. Algebraic relationships for 3-state (ternary) equations are also developed, which show a close relationship to the normal Boolean identities.

This work was originally presented at the British Computer Society Symposium on Logic Design, held at Reading University, July 1967.

Existing techniques for the minimisation of binary combinational switching equations are well established and known. They may be generally classified into two useful categories, namely (*a*) plotting and mapping techniques, as exemplified by Veitch and Karnaugh maps,[1,2] and (*b*) algebraic and numerical techniques, as exemplified by the methods of Quine, McClusky, Flegg, *et. al.*[3,4,5,6]

All the above techniques refer specifically to the minimisation of binary functions whose variables have only two possible values, namely 0 and 1. For ternary functions, whose variables may take three possible values, which will be termed 0, 1, and 2, an extension of the former binary minimisation techniques becomes necessary. Both the binary and the further ternary techniques may, in fact, be regarded as particular cases of the minimisation of s-state functions, where s is any positive integer $\geqslant 2$.

Three-state ternary logic functions

Ternary logic functions will be defined as functions whose one or more input variables $A, B, \ldots, n$ may each take any one of three possible signal levels 0, 1, or 2, and whose output Z may likewise take any one of these three values 0, 1, or 2, under appropriate input conditions. **Fig. 1** below illustrates the difference between normal binary working and such ternary working.

The extension from two to three possible input and output signal levels increases the theoretical number of functions possible in any n-variable system from $2^{2^{(n)}}$ in the binary case, to $3^{3^{(n)}}$ for the ternary case. Thus for any given number of input variables, there is a vastly increased number of ternary functions possible in comparison with the binary case.

Now in the binary case a single equation such as $A\,B\,\bar{C}\,D + A\,\bar{B}\,C\,\bar{D} + \bar{A}\,C\,D$ may express all input conditions that give an output $Z = 1$ (or 0). It is axiomatic that in the absence of these specified input conditions the output Z will be 0 (or 1). It is not therefore necessary to write out both the $Z = 0$ and $Z = 1$ conditions in order to fully specify the system, though as is well appreciated in binary minimisation, one may minimise to fewer terms very much more efficiently than the other, thus making it often desirable to minimise the complement of the function given.

In ternary working, however, the third possible level of every input and output variable complicates the above position. It is possible to write an equation such as:

$$A_0B_1C_2D_2 + A_1C_2D_1 + A_2B_1C_1 = Z_2$$

which specifies that the output Z will be 2 when the four input variables A, B, C, and D have values 0, 1, or 2, as indicated by their attached suffixes. However, absence of these specified input conditions does not, from this single equation, give precise information of the output condition; all it does imply is that the output will not be 2, but whether it is 0 or 1 is not explicit.

Thus *two* equations of the above form are necessary to fully define the ternary system. For example:

$$A_0B_1C_2D_2 + A_1C_2D_1 + A_2B_1C_1 = Z_2$$

and $$A_2B_0C_1D_1 + A_1B_1D_1 + B_2C_1D_0 = Z_1$$

may be given; in the absence of *either* of these specified input conditions then the output will be 0.

Two of the three cases thus have to be realised with appropriate circuits, and combined to give the one output result Z; absence of either of these two states must automatically generate the third output state. In a

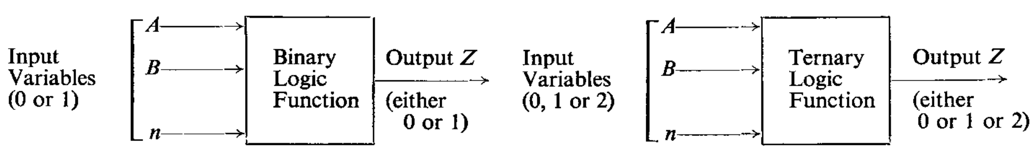

Fig. 1. Binary and ternary logic functions

* *School of Electrical Engineering, Bath University of Technology, Ashley Down, Bristol 7*

manner exactly comparable to the binary case, it may in practice often be more economical to minimise and realise, say, the $Z = 0$ and the $Z = 1$ cases, rather than say the $Z = 1$ and the $Z = 2$ cases.

Ternary circuits which directly realise equations as above have been developed.[7] Such circuits may be employed to synthesise ternary logic equations in a manner very similar to binary synthesis. Such realisation and the algebra as above by which the three output levels of a function are defined, is somewhat dissimilar in concept from that adopted by previous authorities.[8–15] These previous authorities have all expressed a ternary function by one single algebraic expression, often involving complex ternary operators and/or a large number of terms. These complex single expressions do not appear to have any direct method by which they may be minimised.

Minimisation techniques for ternary equations

The minimisation techniques to be detailed will involve the separate minimisation of each individual equation for $Z = 0$, $Z = 1$, and/or $Z = 2$. No combination of say a $Z = 1$ and a dissimilar $Z = 2$ equation will be made, with then a minimisation technique applied to the combined equation. Each will be treated separately. The only exception to this rule is if there is a large group of terms which are common to two equations, though again it is preferable to do a complete separate minimisation, and then look for any common terms in the final minimised expressions. Also since we shall be dealing very largely with the manipulation of the values of each variable of the function, the identification letters A, B, C, etc. of the variables can profitably be dispensed with in many cases. A four-variable function say, $A_1 B_1 C_2 D_0$ can consequently be written quite unambiguously as 1120, in which form it is more amenable to certain search procedures that will be covered later. Care must, however, be exercised in this representation in cases where one or more of the possible variables do not appear; for example a function $A_2 D_2$ must not be written as 22, but rather as $2 - - 2$ to preserve the unwritten variable identification correctly.

In all the minimisation techniques to be considered, there is a fundamental process by which terms are combined and the expression reduced, this process being the search for a variable taking all three possible values 0, 1, and 2. If such a 'complete' variable or 'triplet' can be found, the variable term in question may be deleted.

This feature may be expressed as:

$X_n Y_0 + X_n Y_1 + X_n Y_2 = X_n$, where X_n = any ternary function of one or more variables, and Y = any other ternary variable.

Precise proof of this statement may readily be obtained by applying the fundamental identities given in Appendix A.

This ternary law is the exact counterpart of the binary law:

$X_n Y + X_n \bar{Y} = X_n$, where X_n = any binary function of one or more variables, and Y = any other binary variable.

The above binary law is the underlying basis of all the various two-state minimisation techniques, all of which require searches to be made in some guise for variables which take both values 0 and 1. Minimisation techniques for ternary functions may thus be built up on a similar basis, looking now, however, for ternary terms containing a 'complete' variable 0, 1 and 2.

The ternary minimisation techniques discussed below fall into three general methods, namely (i) simple tabular methods, (ii) graphical methods including mapping, and (iii) algebraic methods utilising the fundamental algebraic identities plus some form of tabulation or correlation procedure. As has already been noted, the minimisation of each individual equation of a ternary system for a particular output condition of 0 or 1 or 2 will be undertaken separately, no combining of these separate equations being possible. Thus the following techniques are applicable equally whether the expression being considered is for output condition Z_0, Z_1, or Z_2, and hence in dealing with the techniques of minimisation, a general expression, viz. $Z = \ldots$, will often be written.

Minimisation of two-variable functions $f(A, B)$ is usually simple, and hence the lowest case we will here consider is three-variable functions $f(A, B, C)$. Any technique discussed for $f(A, B, C)$ is easily relaxed to cover the simpler case of $f(A, B)$ only.

Tabular methods of searching for 'complete' variables 0, 1, and 2, corresponding to the binary tabulation method of Higonnet and Grea,[16] are possible. The tabulations, however, are extremely unwieldy due to the excessive number of columns involved, and hence the graphical and algebraic methods following show greater advantages than any tabular method.

Turning therefore directly to mapping methods, the two-dimensional mapping techniques of binary systems, exemplified by the Veitch and Karnaugh diagrams, may be extended to cater for ternary functions, but with increasing difficulty in the clarity and continuity of their layout, as the number of variables increases. Where, say, Karnaugh maps for binary functions are clear and useful for up to four variables, Karnaugh-type maps for ternary functions become difficult to visualise for over three variables.

The continuity between any one square and any adjacent square that exists in binary Karnaugh maps, including top-to-bottom and end-to-end adjacency, is also difficult to establish in ternary mapping. Several map layouts for three-variable ternary functions may be suggested, but it proves impossible to maintain adjacencies between the $0 \rightarrow 1 \rightarrow 2$ values of all three variables in any two-dimensional layout. The best layout as far as maintaining adjacencies is concerned is as shown in **Fig. 2,** but even this layout has loss of adjacency between the centre $C = 0$ and $C = 1$ regions and the eight surrounding $C = 0$ and $C = 1$ regions.

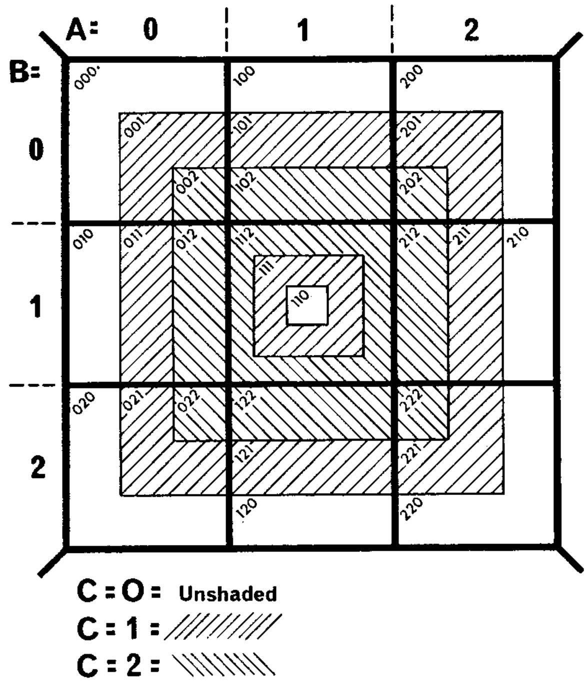

Fig. 2. **Revised 'Veitch/Karnaugh' type diagram for ternary functions of three variables $f(A, B, C)$**

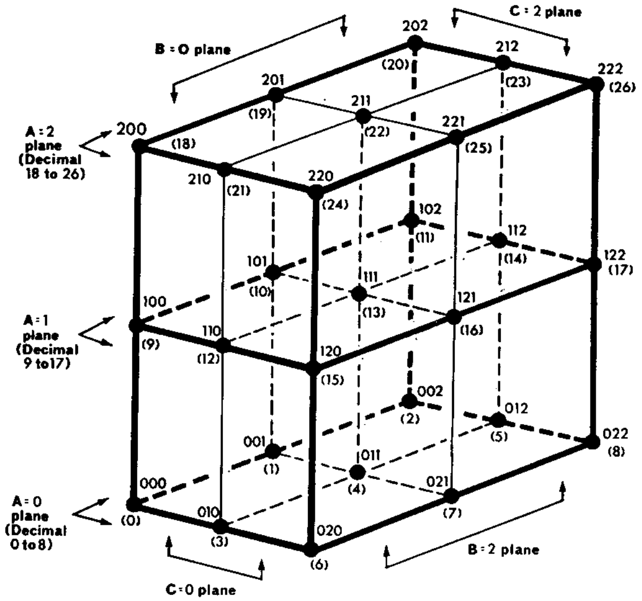

Fig. 3. 3-dimensional representation for ternary functions of three variables $f(A, B, C)$: ('Unit distance' diagram)

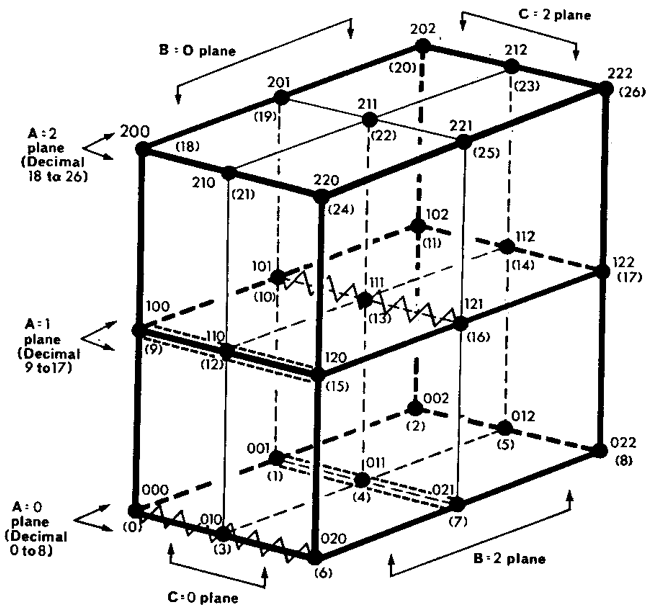

Fig. 4. Plot of $Z = 100 + 011 + 110 + 021 + 120 + 001$ (Shown ====), and also $Z = 000 + 020 + 101 + 121 + 010 + 111$ (Shown ⩘W)

In using such diagrams to minimise a given function, (for example, $A_1B_0C_0 + A_0B_1C_1 + A_1B_1C_0 + A_0B_2C_1 + A_1B_2C_0 + A_0B_0C_1$, which reduces to $A_1C_0 + A_0C_1$), it will be found that the presence of adjacencies in the diagrams is of doubtful assistance in picking out complete variables A or B or $C = 0$, 1, and 2. However, instead of any two-dimensional map-like diagram, a three-dimensional 'unit distance' type diagram for representing three variables is found to be by far the most satisfactory and explicit. It is as shown in **Fig. 3.**

In this diagram the following are points of note:

(*a*) Any straight line of three points represents a function of two variables only.
(*b*) Any complete plane (or surface) of nine points represents a function of one variable only.
(*c*) 'Unit Distance', that is only one function changing by one increment, exists also between any two opposite faces of the cube, i.e. between $A = 0$ and $A = 2$ plane, $B = 0$ and $B = 2$ plane, $C = 0$ and $C = 2$ plane.

Using this 3-dimensional representation to minimise the above example function $Z = 100 + 011 + 110 + 021 + 120 + 001$, the plot indicated by ====== in **Fig. 4** is obtained.

The immediate minimisation of this given function to the two 'lines' representing $A_1C_0 + A_0C_1$ is apparent.

Extending this three-variable minimisation further, suppose the above function was given for $Z = 0$ conditions, and that a further function, say $000 + 020 + 101 + 121 + 010 + 111$ was given for $Z = 1$. Plotting this further function (shown ⩘W in Fig. 4) reveals that it simplifies to $Z = A_0C_0 + A_1C_1$. All remaining points on the diagram must now be for $Z = 2$ output, and examination will immediately reveal that given the above two functions for Z_0 and Z_1, Z_2 is simply $A_2 + C_2$.

In using this three-dimensional diagram for minimisation purposes, the fact that opposite faces of the cube are 'unit distance' apart is of little moment, as three points in line are required to reduce the function by one variable, and three points in line can always be made across a face or internally between faces of the cube, without having to consider any external face-to-face plot.

Considering next the minimisation of equations that are given in AND rather than OR form, that is, algebraically, as products of sums rather than sums of products. Using any mapping or graphical construction, an overlap of all the 'anded' terms must now be sought. The map arrangement shown in Fig. 2 may be used, but again the 3-dimensional representation of Fig. 3 usually affords a preferable presentation of the problem.

Whichever form of plotting for a product of sums equation is used, the plot is not as convenient as that for expressions given as a sum of products, due to the search for overlapping conditions necessary in the former. (This, of course, applies equally to binary systems also.) Two alternatives are, however, available to eliminate this form of plotting, namely:

(i) complementing the complete equation by the extended De Morgan's theorem, (see Appendix A), and plotting the resultant sum of products terms;
(ii) algebraically multiplying out and simplifying the given equation (see Appendix A), with a plotting procedure for final minimisation if necessary.

Turning now from the minimisation of functions of three variables by mapping techniques, to functions of four variables, the difficulty of producing a clear diagrammatic representation begins to become too great, as it does of course with binary systems of five variables.

Two dimensional maps similar to Fig. 2 for a four-variable function $f(A, B, C, D)$ may be proposed, but due to the almost complete lack of adjacencies in the majority of the variables, their use is minimal. Similarly the three-dimensional representation of Fig. 3 may be extended to attempt to cater for four variables, giving the fourth-order hypercube representation shown in **Fig. 5.**

This hypercube representation of a $f(A, B, C, D)$ is shown only partially in Fig. 5, the $D = 0$ points only being shown in full. Any attempt to extend this picture to show all the points of the $D = 1$ and $D = 2$ cube, and all the interconnecting lines between the $D = 0$, $D = 1$, and $D = 2$ cubes becomes completely impractical. It is impossible to 'see' the 81 nodes, 108 lines, 54 surfaces and 12 cubes that are contained in this complete diagram with any degree of clarity and sureness. Thus this fourth-order hypercube representation for $f(A, B, C, D)$ proves completely impractical for minimisation purposes.

To conclude therefore, for functions of up to three ternary variables, a mapping or three-dimensional representation is successful and cannot be bettered. Above three variables, such techniques become impractical, and alternative methods such as those suggested in the following section must be sought.

For the minimisation of ternary functions of four or more variables, recourse to an algebraical/numerical process must be made. As a very first step in this process, the given equation must be expressed in its fully expanded 'sum of products' form, where each product term contains all n variables. For example a term such as $A_0B_0D_2$ of a four-variable system must be expanded into its three constituent 'minterms', namely $A_0B_0C_0D_2 + A_0B_0C_1D_2 + A_0B_0C_2D_2, = 0002 + 0012 + 0022$ for short.

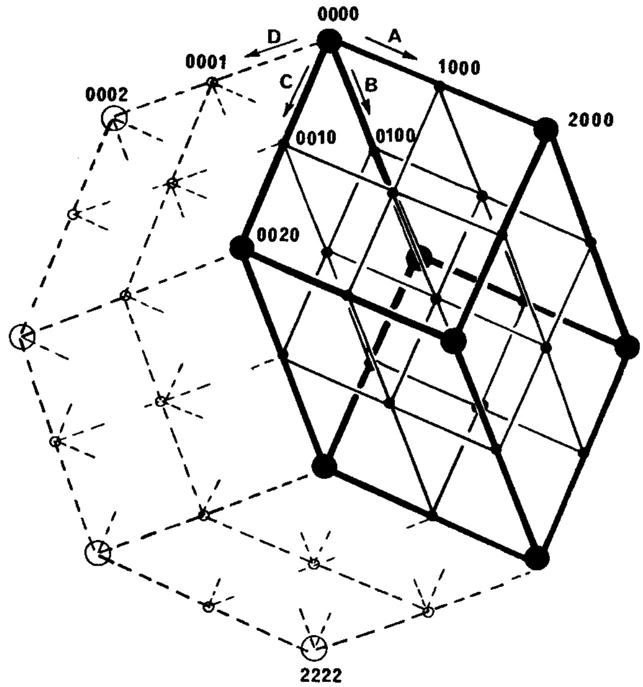

Fig. 5. Hypercube representation of four-variable ternary function $f(A, B, C, D)$

The general minimisation technique is still that of searching for groups of three terms that between them contain one particular variable taking all three values 0, 1, and 2, the remaining $(n - 1)$ variables being identical in the group. When such a group is found, the variable taking the three values 0, 1, and 2, is redundant and may be deleted. The resulting single term is defined as a 'prime implicant'.

There is, however, one serious problem which is not covered by these searches for 'complete' variables in both the binary case and also in the ternary case. It is possible in both cases for some prime implicants to be produced which do not themselves combine to form any further 'complete' variable, and yet these terms still contain some redundancy between them. The given expression is thus NOT ALWAYS COMPLETELY MINIMISED by merely making repeated searches for 'complete' variables, and listing the resulting prime implicants.

This feature was fully appreciated by McClusky, Quine *et al.* in binary working, and often complex subroutines were evolved to deal with the problem. In the ternary minimisation procedure suggested below, it will also be found that the subroutines necessary to sort out any redundancies in the prime implicants are of greater complexity than the relatively simpler searches for complete variables.

As an example of ternary prime implicants which contain between them some redundancy, consider the terms B_1C_1, B_1C_0, A_1C_2, and A_1B_1. These are plotted in 3-dimensional form in **Fig. 6.**

From this diagram it is evident that the A_1B_1 term is in fact redundant, and that minimisation of the stated four terms is given by the first three terms alone. Equally well a 'surface' plus three 'lines' may contain a redundant 'line', for example if 'surface' $C = 0$ had been given instead of the 'line' B_1C_0 in the above example, as of course 'line' B_1C_0 is contained within 'surface' C_0.

A schematic representation of the full reduction procedure is as shown in the tabulation of **Fig. 7.** The main searches for 'complete' variables on the L.H. side require no further comment except that, considered dimensionally, we are collecting together at each step, individual 'points' to form 'lines', individual 'lines' to form 'surfaces', individual 'surfaces' to form 'cubes', and so on, as will be readily appreciated from the diagrams of Figs. 3 or 5.

Considering, however, in more detail the R.H. searches for redundant prime implicant terms.

At the first stage where all possible 'points' have been collected into 'lines', some 'points' may be left over, not combining with any other pairs of terms to form lines. It will, however, be found to be an impossibility for any of such left-over minterms to be redundant, as no one 'point' can be absorbed either by any other combination of left-over 'points', or by any combination of left-over 'points' plus established 'lines'. Reference to say the 3-dimensional figure first given in Fig. 3 will by trial verify this statement. Thus all minterms left over after the first search for complete variables may be printed out immediately as being necessary terms in the minimisation procedure, i.e. are prime implicants that must appear in the final minimised solution.

Following the next L.H. search, however, when 'lines' are combined to form 'surfaces', a technique for the elimination of possible redundant 'lines' becomes necessary. It is impossible for any 'line' to be left over that is wholely included within any single established 'surface' term, but a comparison of each left-over term with three other terms may reveal that the left-over

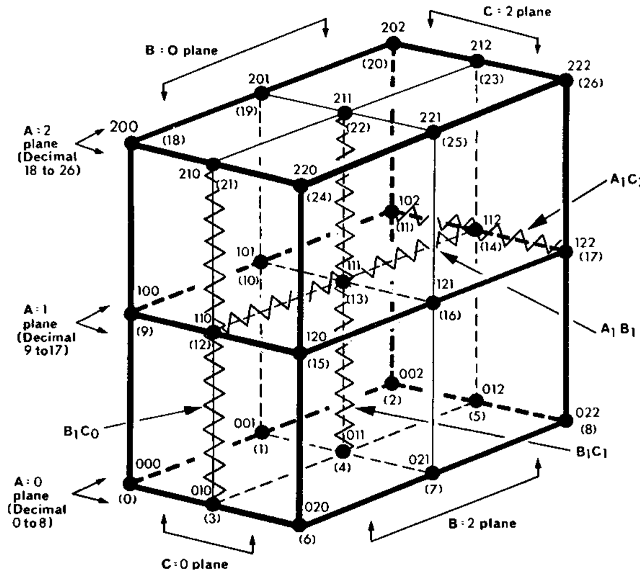

Fig. 6. Plot of $B_1C_1 + B_1C_0 + A_1C_2 + A_1B_1$

term is completely contained by the three terms, and is thus redundant. One such case has been illustrated in Fig. 6.

On examining all such cases, it will be found that the '—' variable of the redundant term takes the three possible values 0 and 1 and 2 in the triplet of the three other terms. The remaining $(n - 1)$ variables of the redundant term are each identical to the corresponding $(n - 1)$ variables of the triplet, where '—' in any of the triplet terms is taken as being equal to 0 or 1 or 2 for the purpose of this identity comparison.

This comparison technique will detect all such redundant prime implicant terms. Thus the complete step-by-step procedure for the minimisation of any ternary function of n variables, given schematically in Fig. 7, is as tabulated in Appendix B, a decimal number order of tabulation of the given minterms being suggested to aid the initial search for 'points' combining to form 'lines'.

As an illustration of the above algebraic minimising technique, consider the following four-variable expression:

$$Z = A_2C_1D_1 + B_0D_2 + A_1B_0C_2 + A_1B_0D_0 + A_1B_0C_0D_1 + A_1C_1D_1 + B_2D_1 + A_1B_1D_1.$$

(1) Expanding into sum of minterm form:

$A_2C_1D_1$	+	B_0D_2	+	$A_1B_0C_2$	+	$A_1B_0D_0$	+	$A_1B_0C_0D_1$	+	$A_1C_1D_1$	+	B_2D_1	+	$A_1B_1D_1$
2011	+	0002	+	1020	+	1000	+	1001	+	1011	+	0201	+	1101
2111		1002		1021		1010				1111		1201		1111
2211		2002		1022		1020				1211		2201		1121
		0012										0211		
		1012										1211		
		2012										2211		
		0022										0221		
		1022										1221		
		2022										2221		

(2 and 3) Adding together 'values' and tabulating in resultant decimal number order:

DECIMAL TOTAL =	0	1	2	3	4	5	6	7	8
List of all minterms:	—	1000	1010 0002 1001	1020 1002 0012 1020* 1011 0201 1101	2002 1012 1021 1111 1201 1111* 0022 0211 2011	2012 1022 1211 2201 1121 1022* 1211* 0221 2111	2022 2211 1221 2211	2221	—

(4) Delete all duplications in above tabulation, (= those indicated by *).

(5) Compare each term in above tabulation with all other terms in next two columns, looking for complete variables:

1000	1000	0002	0002	1010
1001	1010	1002	0012	1011
1002	1020	2002	0022	1012
100–	10–0	–002	00–2	101–
1001	1001	1020	1002	0012
1101	1011	1021	1012	1012
1201	1021	1022	1022	2012
1–01	10–1	102–	10–2	–012
1011	0201	0201	1101	2002
1111	0211	1201	1111	2012
1211	0221	2201	1121	2022
1–11	02–1	–201	11–1	20–2
1021	1201	0022	0211	2201
1121	1211	1022	1211	2211
1221	1221	2022	2211	2221
1–21	12–1	–022	–211	22–1
0221	2011			
1221	2111			
2221	2211			
–221	2–11			

(6) Tabulating the above 'lines' in '—' order:

xxx —	xx — x	x — xx	— xxx
100— 101— 102—	10—0 00—2 10—1 10—2 02—1 11—1 20—2 12—1 22—1	1—01 1—11 1—21 2—11	—002 —012 —201 —022 —211 —221

(7) Print out any minterms not combining in (5), = NONE.

(8) Comparing each term in each column with all other terms in the same column, looking for complete variables:

100–	10–0	00–2	10–1
101–	10–1	10–2	11–1
102–	10–2	20–2	12–1
10––	10––	–0–2	1––1
02–1	1–01	–002	–201
12–1	1–11	–012	–211
22–1	1–21	–022	–221
–2–1	1––1	–0–2	–2–1

(9) Tabulating the above 'surfaces' in '—' order, deleting any duplications:

xx——	x—x—	—xx—	—x—x	x——x	——xx
10––			–0–2 –2–1	1––1	

(9*a*) Left-over 'line' terms of (6) that do not contribute in forming 'surfaces' = 2 — 11 only.

Now no triplet of terms before the last minimisation, with 0, 1, and 2 in the second variable position and complete agreement of all three other variables, is present.

∴ This left-over term is an irredundant prime implicant in the final minimisation.

(10) No further complete variables are available in tabulation (9). This therefore is the final minimisation, as these terms by themselves contain no redundancies. Therefore given expression simplifies to:

$$Z = 10\text{––} + \text{–}0\text{–}2 + \text{–}2\text{–}1 + 1\text{––}1 + 2\text{–}11$$
$$= A_1B_0 + B_0D_2 + B_2D_1 + A_1D_1 + A_2C_1D_1.$$

NOTE: Function must first be expressed in 'sum of minterms' form, e.g. $Z = f_1 + f_2 + f_3 + \ldots$, where f_1, f_2, f_3, etc., each contain all n variables.

1. Establish and list all unique 'points'.

$= \Sigma ABCDE$

2. Look for and list all combinations of 'points' that give a complete variable 0, 1, and 2, = establish all unique 'lines'. → 2(*a*) Print out all 'points' not combining to form 'lines' = *irredundant terms.*

↓ e.g. $A_0B_1{}^*D_2E_0$, etc.

3. Look for and list all combinations of 'lines' that have a further complete variable 0, 1, and 2, = establish all unique 'surfaces'. → 3(*a*) Compare all 'line' terms not combining to form 'surfaces' for redundancies. Delete all such redundancies and print out all remaining terms = *irredundant terms.*

↓ e.g. $A_0B_1{}^*D_2{}^*$, etc.

4. Look for and list all combinations of 'surfaces' that have a further complete variable 0, 1, and 2, = establish all unique 'cubes'. → 4(*a*) Compare all 'surface' terms not combining to form 'cubes' for redundancies. Delete all such redundancies and print out all remaining terms = *irredundant terms.*

↓ e.g. ${}^*B_1{}^*D_2{}^*$, etc.

5. Look for and list all combinations of 'cubes' that have a further complete variable 0, 1, and 2, = establish all unique fourth order 'hypercubes'. → 5(*a*) Compare all 'cube' terms not combining to form 'hypercubes', for redundancies. Delete all such redundancies and print out all remaining terms = *irredundant terms.*

↓ e.g. ${}^*{}^*{}^*D_2{}^*$, etc.

6. Repeat this complete variable search for a total of n times where n = number of variables in the original terms, or when no further complete variables are found. → 6(*a*) Repeat this redundancy search between each step, printing out all nonredundancies = *irredundant terms.*

↓

Final Minimum Solution of original Function = Sum of all the irredundant terms of the R.H. sub-programs, = sum of all these Prime Implicants.

Fig. 7. Reduction procedure for ternary functions of *n* variables

The above minimisation technique will thus be seen to be a large number of individually simple searches for specific arrangements and correspondence of variable values. It is thus a relatively simple exercise for any digital computer.

Conclusions

The minimisation techniques discussed above show a close relationship to the more familiar techniques associated with binary functions. The same problems, such as the inability to map multi-variable functions, occur more forceably in ternary working than in binary, but these problems are fundamentally identical.

The numerical techniques as adopted above for the detection and hence elimination of redundant prime implicants produced during a numerical minimisation procedure, can be applied equally to detecting redundant binary prime implicants during a Quine or McClusky binary minimisation procedure. As only the two values 0 and 1 are present in the binary case, the procedure for detecting redundant prime implicants is somewhat simplified.[17]

Both the ternary and the binary minimisation cases are in fact particular cases of the minimisation of m-valued logic functions. The numerical minimisation technique for m-valued functions requires repeated searches for variables taking the values 0, 1, 2, . . ., $(m-1)$, with the elimination of redundant prime implicants undertaken in a manner similar to the binary and ternary cases.[17]

References

1. Veitch, E. W. (1952). A Chart Method for Simplifying Truth Functions, Proc. Assn. for Computing Mach. Conf., May 1952, pp. 127–133.
2. Karnaugh, M. (1953). A Map Method for Synthesis of Combinational Logic Circuits, *Trans. A.I.E.E.*, Vol. 72, pp. 593–599.
3. Quine, W. V. (1955). A Way to Simplify Truth Functions, *American Mathematical Monthly*, Vol. 62, pp. 627–631.
4. McCluskey. E. J. (1956). Algebraic Minimisation and the Design of Two-Terminal Contact Networks, M.I.T. Thesis.
5. Staff of Harvard Computation Centre (1951). *Synthesis of Electronic Computing and Control Circuits*, Harvard U.P.
6. Flegg, H. G. (1959). The Manipulation and Minimisation of Boolean Switching Functions, College of Aeronautics Thesis.
7. Hurst, S. L. (1968). Semiconductors Circuits for Three-State Logic Applications, *Electronic Engineering*, Vol. 40, pp. 197–202, and pp. 256–259.
8. Post, E. L. (1921). Introduction to a General Theory of Elementary Propositions, *Amer. Journal of Mathematics*, Vol. 43, pp. 163–185.

9. Bernstein, B. A. (1924). Modular Representations of Finite Algebras, Proc. 7th International Congress of Mathematicians, Toronto, Vol. 1, pp. 207–216.
10. Rosenbloom, P. C. (1942). Post Algebras: Postulates and General Theory, *Amer. Journal of Mathematics*, Vol. 64, pp. 167–188.
11. Webb, D. L. (1953). Generation of any *N*-Valued Logic by one Binary Operation, *Proc. Nat. Academy Sciences*, Vol. 21, pp. 252–254.
12. Rohleder, H. (1954). Three Valued Calculus of Theoretical Logics and its Application to the Description of Switching Circuits which consists of Elements of Two States, *Zeitschrift für Augewandte Mathematik und Mechanik*, Vol. 34, pp. 308–311.
13. Chen, C. Y., and Lee, W. H. (1957). Several Valued Switching Circuits, *Trans. A.I.E.E.*, Vol. 25, Pt. 1, pp. 278–283.
14. Vacca, R. (1957). A Three-Valued System of Logic and its Application to Base Three Digital Circuits, UNESCO/NS/ICIP/G.2.14.
15. Mühldorf, E. (1958). Ternäre Schaltalgebra, *Arch. Elektrischen Übertragung*, pp. 138–148 (in German).
16. Higonnet, R., and Grea, R. (1958). *The Logical Design of Electrical Circuits*, McGraw Hill.
17. Hurst, S. L. (1966). Boolean Minimisation Techniques, *I.E.E. Electronics Letters*, Vol. 2, No. 8, p. 291.

Appendix A

An extension of binary algebraic identities to ternary functions

The fundamental identities associated with two-state Boolean operations may be extended to cover three-state (ternary) working, as shown below. A close parallel will be seen to exist between established two-state identities and these three-state identities, but the presence of the third possible state in the latter often makes the application of these identities more complex in manipulation and minimisation problems.

Elementary propositions

$$(a)\ \begin{cases} A_0 + A_0 = A_0 \\ A_1 + A_1 = A_1 \\ A_2 + A_2 = A_2 \end{cases}$$

$$(b)\ \begin{cases} A_0 . A_0 = A_0 \\ A_1 . A_1 = A_1 \\ A_2 . A_2 = A_2 \end{cases}$$

$$(c)\quad A_0 + A_1 + A_2 = 1$$

$$(d)\ \begin{cases} A_0 . A_1 = 0 \\ A_0 . A_2 = 0 \\ A_1 . A_2 = 0 \\ A_0 . A_1 . A_2 = 0 \end{cases}$$

$$(e)\ \begin{cases} \bar{A}_0 = A_1 + A_2 \\ \bar{A}_1 = A_0 + A_2 \\ \bar{A}_2 = A_0 + A_1 \end{cases}$$

$$(f)\ \begin{cases} A_0 + 1 = 1 \\ A_1 + 1 = 1 \\ A_2 + 1 = 1 \end{cases}$$

$$(g)\ \begin{cases} A_0 . 1 = A_0 \\ A_1 . 1 = A_1 \\ A_2 . 1 = A_2 \end{cases}$$

$$(h)\ \begin{cases} A_0 + 0 = A_0 \\ A_1 + 0 = A_1 \\ A_2 + 0 = A_2 \end{cases}$$

$$(j)\ \begin{cases} A_0 . 0 = 0 \\ A_1 . 0 = 0 \\ A_2 . 0 = 0 \end{cases}$$

In the above list of elementary propositions, the numbers '0' and '1' occurring as a operator or as a resultant may be interpreted as '0' = nothing or 'zero class', and '1' = whole complement or 'universal class' in the usual manner.

Associative and commutative laws

$$(k)\quad (A + B) + C = A + (B + C)$$
$$(l)\quad (A . B) . C = A . (B . C)$$
$$(m)\quad A + B = B + A$$
$$(n)\quad A . B = B . A$$

Distribution laws

$$(p)\quad A . (B + C) = A . B + A . C$$
$$(q)\quad A + B . C = (A + B) . (A + C)$$

Extended De Morgan's theorem

$$(r)\quad \overline{(A + B + \ldots)} = \bar{A} . \bar{B} . \cdots$$
$$(s)\quad \overline{(A . B . \cdots)} = \bar{A} + \bar{B} + \cdots$$

As an example of these final complemented identities, the complement of the function $(A_2 + B_2 + C_0)$ is:

$$\overline{(A_2 + B_2 + C_0)} = \bar{A}_2 . \bar{B}_2 . \bar{C}_0,$$
$$= (A_0 + A_1) . (B_0 + B_1)\ (C_1 + C_2),$$

which may be multiplied out if desired to give the following result in minterm form:

$$A_0B_0C_1 + A_0B_1C_1 + A_1B_0C_1 + A_1B_1C_1 + A_0B_0C_2 + A_0B_1C_2 + A_1B_0C_2 + A_1B_1C_2$$

The above identities may be employed to minimise ternary algebraic equations in a manner closely analogous to binary working. For example, consider the ternary equation:

$$(A_1 + B_1 + C_0) . (A_2 + B_1 + C_0) . (A_0 + B_2 + C_2).$$

Multiplying out and applying the ternary identities we obtain:

$$(A_1 + B_1 + C_0)(A_2 + B_1 + C_0)(A_0 + B_2 + C_2),$$

$$= (A_1A_2 + A_1B_1 + A_1C_0 + B_1A_2 + B_1B_1 + B_1C_0 + C_0A_2 + C_0B_1 + C_0C_0)(A_0 + B_2 + C_2),$$

$$= (A_1B_1 + A_1C_0 + B_1A_2 + B_1 + B_1C_0 + C_0A_2 + C_0B_1 + C_0)(A_0 + B_2 + C_2),$$

$$= A_0A_1B_1 + A_0A_1C_0 + A_0B_1A_2 + A_0B_1 + A_0B_1C_0 + A_0C_0A_2 + A_0C_0B_1 + A_0C_0$$

$$+ B_2A_1B_1 + B_2A_1C_0 + B_2B_1A_2 + B_2B_1 + B_2B_1C_0 + B_2C_0A_2 + B_2C_0B_1 + B_2C_0$$

$$+ C_2A_1B_1 + C_2A_1C_0 + C_2B_1A_2 + C_2B_1 + C_2B_1C_0 + C_2C_0A_2 + C_2C_0B_1 + C_2C_0.$$

Deleting all further inadmissable ('zero') terms in the above, .e. $A_0A_1B_1$, $A_0A_1C_0$, etc., the remaining terms are:

$$A_0B_1 + A_0B_1C_0 + A_0C_0B_1 + B_2A_1C_0 + B_2C_0A_2 + B_2C_0 + A_0C_0 + C_2A_1B_1 + C_2B_1A_2 + C_2B_1,$$

$$= A_0B_1(1 + C_0 + C_0) + B_2C_0(A_1 + A_2 + 1) + A_0C_0 + B_1C_2(A_1 + A_2 + 1),$$

$$= A_0B_1(1) + B_2C_0(1) + A_0C_0 + B_1C_2(1),$$

$$= A_0B_1 + B_2C_0 + A_0C_0 + B_1C_2.$$

This final result cannot further be minimised by algebraic means. However, precisely as in binary minimisation techniques, such ternary algebraic reductions do not always guarantee a minimum solution, and one or more of the final terms may still be redundant. Graphical or numerical search procedures should be applied to check for such redundancies.

The identities and algebraic manipulations discussed above equally well may be extended to higher-valued functions than the ternary case here considered.

Appendix B

Minimisation procedure for function $Z = \Sigma\ ABCDE \ldots$

1.1 Expand each term of given OR function into all its minimum polynomials or 'minterms', = expand to give all 'points'. (e.g. $A_1B_1C_1E_1 = A_1B_1C_1D_0E_1 + A_1B_1C_1D_1E_1 + A_1B_1C_1D_2E_1 = 11101 + 11111 + 11121$ for convenience).

1.2 Add together the values of each of the n variables in each minterm and hence establish an order of the minterms in accordance with these decimal number totals (e.g. 21120 = '6').

1.3 Tabulate all minterms in columns, the column order being these decimal totals 0, 1, 2, . . ., $2n$:
e.g.

0	1	2	3	——	$2n$
—	01000	—	11100 02010		

1.4 Delete any duplications in each column, = establishment of all unique points.

1.5 Compare each term in each column with all terms in the next two higher columns, looking for one variable taking the values 0, 1, and 2 in the three terms being compared, with all other $(n - 1)$ variables identical. Note: Any one term may contribute up to n times in this correspondence search. (This is looking for 3 'points' → 1 'line'.)

1.6 Tabulate out each 'line' term so found in columns corresponding to the position of the complete variable, with '—' or some other chosen symbol or number other than 0 or 1 or 2 in place of the complete variable; (= tabulation of all unique 'lines'; suggest the symbol '—' is used to represent the complete variable 0 or 1 or 2).

1.7 Print out all minterms not combining to form any 'line', = necessary IRREDUNDANT PRIME IMPLICANTS.

1.8 Compare each 'line' term of tabulation (6) with all other terms in the same column, looking for any one variable in the terms being compared that takes the value 0, 1, and 2, with all other $(n - 1)$ terms identical; (= looking for 3 'lines' → 1 'surface').

Note: Any one term may contribute up to $(n - 1)$ times in this search.

1.9 Tabulate out each 'surface' so found in columns corresponding to the positions of the complete variables, with '—' or other chosen symbol in place of the second complete variable. Delete any duplications in each column; (= establishment of all unique surfaces). Then:

(*a*) Complete the subroutine program (2) detailed below, to eliminate any redundant 'line' terms left over after this collection together of 'lines' into 'surfaces'.

(*b*) Print out all remaining 'line' terms following this elimination check, = necessary IRREDUNDANT PRIME IMPLICANTS.

1.10 Compare each 'surface' term of previous tabulation with all other terms in the same column, looking for any one variable in the terms being compared that takes the value 0, 1 and 2, with all other $(n - 1)$ terms identical; (= looking for 3 'surfaces' → 1 'cube').

Note: Any one term may contribute up to $(n - 2)$ times in this search.

1.11 Tabulate out each 'cube' so found, in columns corresponding to the positions of the complete variables, with '—' or other chosen symbol in place of the third complete variable. Delete any duplications in each column, (= establishment of all unique cubes). Then:

(*a*) Repeat subprogram (2) below to sort out redundant 'surface' terms. Print out all non-redundant 'surface' terms = necessary IRREDUNDANT PRIME IMPLICANTS.

1.12 Repeat (10) and (11) until no more complete variables 0, 1 and 2 are found in this search for triplets. Print out all IRREDUNDANT PRIME IMPLICANTS at end of each subprogram. Procedure to finish on a subprogram routine.

2. *Subroutine program between above Steps* 9 *and* 10, 11 *and* 12, *etc.*

2.1 For each term left over after the completion of each main minimisation search, look at ALL terms prior to this last minimisation search, looking for a triplet of terms with 0, 1, and 2 in the position of any '—' variable of the left-over term.

2.2 If one or more such triplet of terms can be found, check agreement of all remaining $(n - 1)$ variables of each term of each triplet with the left-over term, where (i) any additional '—' variable in the left-over term must match exactly with a '—' variable in each of the triplet terms, and (ii) all remaining 0 or 1 or 2 variables of the left-over term must match exactly with the corresponding variable in each of the triplet terms, where '—' in any triplet term is taken as equal to 0 or 1 or 2 as desired.

2.3 If such full agreement of all these $(n - 1)$ variables of a triplet and the left-over term is found, the left-over term is redundant and should be deleted.

2.4 If no triplets or matching of triplet variables can be found, the left-over term is irredundant and must therefore be printed out as a necessary IRREDUNDANT PRIME IMPLICANT.

(*Note*: Having found redundant and deleted a left-over term, this term is no longer available in the list of ALL terms prior to the last minimisation, see 2.1 above. Thus if procedures 2.1 to 2.4 above have to be repeated for further left-over terms, any deleted term is no longer available in these subsequent search procedures.)

Book Review

Machine Intelligence 3. (ED.) D. MICHIE, 1968; 405 pages. (*Edinburgh University Press*, 70s.)

The first attempts to get machines seeking proofs for mathematical assertions were aimed at putting mathematicians out of business and remedying Fermat's deplorable carelessness. Ten years and a few pages of college mathematics later the day nevertheless looks not far off when theorem-proving will indeed be a workaday occupation for computers in banks and universities, but not to lay Goldbach's ghost. The light is slowly dawning that even a modest inferential capacity would be an immense improvement on today's lumpen responses, and that mathematics is not the only illogical human activity that lends itself to analysis in the terms of current logic.

In five papers of this book leading participants in the theorem-proving field write about techniques for proof-seeking, especially J. A. Robinson and others about developments of Robinson's 'resolution principle'. To make a comparison with another field, their work is like the development of techniques for dealing with simultaneous linear constraints. For applications we must await the next instalment (at least). This comparison is likely to be justified when proof-seeking acquires a similarly central position to that now occupied by linear programming.

The analogy doesn't stop here. Operational research began as the ragbag for unclassifiable applications of mathematics, and it spawned linear programming as its earliest specific methodology. For OR the search for self-identification is now over (or just too boring) and its boundaries have hardened. The focus for the unclassifiable has shifted from the analysis of corporate behaviour to the analysis of individual behaviour. The mantle of OR seems to have fallen on artificial intelligence (AI), although the shift is by no means complete, witness here Varshavsky's survey of recent Russian work in Collective Behaviour and Control.

Theorem-proving started as an application-study and is becoming a foundational tool. Another less surprising such tool is the exploration of trees and graphs. Also, reaching out unsurely towards the proof-seeking techniques, is the business of formulating in logic topics that might one day yield to mechanical inference-making, for example Laski's and Park's discussions of data-structures. These shade off from logic into programming because programming languages seem destined to evolve towards logic. AI can take a lot of the credit for this. It has always excelled at spot-lighting the deficiencies that most of us merely suffer inarticulately (see Burstall's 'Alternative Expressions', Foster's 'Assertions').

As ever more tricks are found for introducing implicit forms of description instead of the explicit forms forced on us by strictly algorithmic languages, something has to be done to clean up the mess, and logicians are technology's sanitary inspectors.

If AI has inherited some of the angst of OR it has also inherited a sympton—polarisation between methodological preoccupations and undigested engineering descriptions. This book bridges the gap with some thoughtful case-studies, especially Amarel exhibiting the effect of alternative formulations of a problem (missionaries and cannibals).

One serious study for its own sake is the automatic English parser of Thorne *et al.* that does not rely on a complete dictionary of words encountered. Language processing is another application area that has won independence. Perhaps AI is destined to remain fuzzy because each study that becomes well-defined claims autonomy. Some practitioners might draw the bounds so tight as to include only one piece of work in this book—Hilditch's automatic inspection of photographs of chromosomes.

Fortunately this volume represents no such narrow view. The preface acknowledges the difficulty of assimilating 'the interconnections of such a ramifying field of subject matter'. Reading the book brings home how important doing just that is going to be as AI crystallises out into techniques and application-areas the shapes of which are not now predictable.

P. J. LANDIN (London)