VISOR, *QUOT*, and *REM*, subject to the restriction $REM < DIVISOR$. The value for *QUOT* was then divided by 2^k, where k was an integer chosen randomly in the interval $0 \leqq k \leqq 31$. The dividend was then computed from equation (9), and the division of *DIVIDEND* by *DIVISOR* begun. The resulting quotient and remainder were compared to the known values, and diagnostic information was printed in case of any disagreement. Over 18 million separate tests using several different random number generators were made of the division algorithm at a rate of 100,000/min. The bounds on the difference between the true and tentative quotients given in eqs. (18) and (19) were verified, and the algorithm is known to be correct.

5. A System/360 Implementation of the Division Algorithm

To illustrate the coding of the division scheme on an actual machine, a code sequence is given in Figure 3 which will perform the division algorithm on an IBM System/360 computer. It contains one additional test not shown in Figure 2, if the divisor has a positive arithmetic representation; if $DIVIDEND < M$ (that is, the high order part of the dividend is zero), then the division process may be skipped. Because all System/360 fixed point addition instructions (as well as the logical OR-instruction) change the condition code [2], the quantity $4R + A$ must be computed by first forming $4R$ and testing it for overflow, and later adding A, which cannot then cause an additional overflow.

Acknowledgments. The author thanks M. D. McIlroy and the referee for several helpful suggestions concerning the presentation of the material.

RECEIVED APRIL, 1967; REVISED DECEMBER, 1967

REFERENCES

1. A multiple precision floating point subroutine package for System/360. Tech. Memo. No. 18, Stanford Linear Accelerator Center Comput. Group, Stanford U., Stanford, Cal.
2. IBM System/360 principles of operation. File No. S360-01, Form A22-6821, IBM.

Generating Prime Implicants Via Ternary Encoding and Decimal Arithmetic

D. L. DIETMEYER AND J. R. DULEY
University of Wisconsin, Madison, Wisconsin*

Decimal arithmetic, ternary encoding of cubes, and topological considerations are used in an algorithm to obtain the extremals and prime implicants of Boolean functions. The algorithm, which has been programmed in the FORTRAN language, generally requires less memory than other minimization procedures, and treats DON'T CARE terms in an efficient manner.

KEY WORDS AND PHRASES: prime implicants, extremal, switching function, minimization, cubical complexes, ternary encoding
CR CATEGORIES: 3.24, 6.1

Almost all of the machine oriented minimization algorithms proposed reduce to two disjoint parts: the first part is a process for generating information which includes the minimum expression of a Boolean function, while the second part analyzes that information and selects the minimum expression of the Boolean functions. There are two ways in which digital computer programs that are designed to execute the algorithms fail. First, most generation processes create large lists of intermediate information, and the finite high speed memory of the digital computer is not capable of storing these large lists. Second, the information may be in such a complex form that it is not practical to pay for the computer time needed to extract the minimum solution. In the second case, the designer may have to be satisfied with a solution which is not absolutely minimal. However, when the computer fails during the generation process, the problem must be rejected before reaching the selection stage, a stage which might be completed with relative ease.

This note attempts to alleviate this more serious problem through a refined and generalized version of a procedure pioneered by Urbano-Mueller [1], Harris [2], and Prather [3]. The algorithm presented partially integrates the problems of generation and selection, and is constructed

* Department of Electrical Engineering.

to avoid the creation of very large lists of intermediate information. The method does not, in general, produce a minimum solution, but a minimum solution may be extracted from the results using existing algorithms. In addition, the algorithm developed is a decimal procedure and uses arithmetic operations throughout instead of the logical operations which are usually available only in the machine language of large binary machines.

The algorithm is topological in nature and is based on the following properties of the familiar Boolean n-cube which is used to express a Boolean function of n variables [5, 6].

Property 1. An n-cube contains 2^n vertices each of which has a unique position in the n-cube,

Property 2. Every vertex in an n-cube is adjacent to n vertices of the n-cube.

Property 3. The distance between any two vertices in an n-cube must be less than or equal to n.

Property 4. A j-cube that contains vertex V^0, may be formed by combining j $(j-1)$-cubes which contain V^0 with the proper vertex, $V_1{}^j$. V^0 and $V_1{}^j$ are distance j apart.

From basic number system theory any ternary integer of n trits may be represented as

$$I = \sum_{i=0}^{n-1} d_i 3^i, \qquad d_i = 0, 1 \text{ or } 2. \tag{1}$$

Any Boolean term may be represented by the ternary integer obtained by replacing all variables in the term by 1's, all complements by 0's, and missing variables by 2's. Evaluating the power series of eq. (1) by performing decimal arithmetic gives the decimal equivalent of any ternary integer. The subcube $A\text{–}\bar{C}D$ is thus represented by

$$(1201)_3 = (46)_{10}.$$

In brief, the algorithm of this note selects a vertex V^0 from the ON-array and analyzes the surrounding cubical structure in the K-array (ON and DON'T CARE vertices) to find the prime implicants that cover the selected vertex. This process is repeated as necessary until each member of the ON-array has been covered. Let U^k denote the set of vertices that are distance k from V^0. Each member of U^k is denoted by V_i^k.

THEOREM 1. *All members of U^1 differ from V^0 by a power of three, i.e. $|V^0 - V_i^1| = 3^n$ for n a positive integer.*

PROOF. All V_i^1 must be adjacent to V^0 by the definition of U^1 and exactly one coordinate position must differ between the binary representations of the two vertices. Let the difference occur in the ith position. If $V^0 = (d_n \cdots \bar{d}_i \cdots d_1)_3$ and $V_j^1 = (d_n \cdots d_i \cdots d_1)_3$, then

$$|V^0 - V_i^1| = (0 \cdots 010 \cdots 0)_3 = 3^{i-1}. \qquad \text{Q.E.D.}$$

U^1 consists of all V_i^1's, the adjacent vertices that form 1-cube covers of V^0, and U^1 is generated by searching the K-array for all vertices that differ from V^0 by a power of three. If U^1 contains d members, a d-cube is the largest possible cover of V^0 (Property 2).

A vector approach is useful in determining the vertices that complete larger cubes. The d vertices of U^1 may be used to form d vectors ΔV_i by subtracting V^0 from each V_i^1.

$$\Delta V_i = V_i^1 - V_0 \quad \text{or} \quad V_i^1 = V^0 + \Delta V_i. \tag{2}$$

THEOREM 2. *Adding j ΔV_i's to V^0 complements j coordinates and locates a vertex $V_1{}^j$ which is distance j from V^0.*

PROOF. ΔV_i is a power of three as a consequence of eq. (2) and Theorem 1. Thus, adding ΔV_i to V^0 can change only one variable position. Adding one ΔV_i complements one variable since it locates an adjacent vertex, V_i^1. Each ΔV_i complements a unique variable position since each ΔV_i is unique. Thus, if another ΔV_i is added to the sum, another variable is complemented and the distance from V^0 is increased by one. By finite induction the theorem results. Q. E. D.

With the aid of the ΔV_i's, all the vertices of the hypothetical d-cube cover of V^0 may be located by adding each of the 2^d possible combinations of the ΔV_i's to V^0. These vertices form sets U^1 to U^n. After the adjacent vertices are found by Theorem 1 and the ΔV_i's are calculated, then the V_i^1's are placed side by side forming d columns (set U^1). From then on U^{j+1} is found by applying the following Generation Rules to U^j.

Generation Rules:

(1) The members of the first column of U^{j+1} are found by adding ΔV_1 to each member of every column to the right of the first column in U^j.

(2) The members of the second column of U^{j+1} are found by adding ΔV_2 to each member of every column to the right of the second column in U^j.

. .
. .
. .

(k) The members of the kth column of U^{j+1} are found by adding ΔV_k to each member of every column to the right of kth column in U^j.

This iterative process is self-terminating since each set has one less column than the last set.

The foregoing vector approach shows only how each vertex of a hypothetical d-cube may be calculated. When a vertex in U^2 is calculated, it is known to be a vertex that will complete a 2-cube, but it must be a member of the K-array before it actually does complete a 2-cube cover of V^0. In general, one or more vertices of the hypothetical d-cube cover of V^0 are not a part of the K-array and the hypothetical d-cube is not an actual cover of V^0. To distinguish between vertices which complete hypothetical and actual covers of V^0, defined $U_c{}^i$ as the subset of U^j which consists of all $V_i{}^j$'s that complete actual j-cube covers of V^0. U_{nc}^j contains the remainder of U^j or the $V_i{}^j$'s that do not complete actual j-cube covers of V^0 because they themselves or other vertices are absent from the K-array. Members of $U_c{}^j$ are easily recognized by Theorem 3.

THEOREM 3. *If a vertex in U^j is adjacent to j members of U_c^{j-1} and is a member of the K-array, it is a member of U_c^j.*

PROOF. If V_i^j is to form a j-cube cover of V^0, it must be adjacent to j vertices of set U^{j-1} by Properties 2 and 3. If these j adjacent vertices are members of U_c^{j-1} and if V_i^j is a member of the K-array, then by Property 4, V_i^j completes a j-cube cover of V^0 and is a member of U_c^j. Q.E.D.

If, after generating set U^{j-1}, U_c^{j-1} is found to contain less than j members, Theorem 3 states that no j-cube exists and U^j need not be generated.

An algorithm for generating the cube(s) that cover any vertex of the ON-array may be summarized:

Step 1. Select a V^0 from the ON-array.

Step 2. Use Theorem 1 to generate U^1 by searching for vertices in the K-array which differ from V^0 by a power of three.

Step 3. Calculate the set of ΔV_i vectors from eq. 1.

Step 4. Use the ΔV_i vectors to calculate the next set of vertices from the known set using the Generation Rules. Those members of this new set which are contained in the K-array and comply with Theorem 3 complete and represent cubes that cover V^0.

Step 5. If U_c^{j-1} contains less than j vertices, stop generation. If not, return to Step 4.

One or more of the cubes which cover V^0 (implicants) must be included in the minimum expression of the Boolean function. As a first step in the selection of the cubes which make up the ultimate minimum cover, prime implicants must be recognized where a *prime implicant* is a cube of the K-array not included in any other cube of the K-array. The determination of prime implicants reduces to detecting all uncovered cubes. Property 4 indirectly states that the j $(j-1)$-cubes that cover V^0 are a part of a hypothetical j-cube cover of V^0. If that j-cube actually exists as a cover of V^0, then all j $(j-1)$-cubes are not prime implicants since they are covered by the j-cube. If all cubes so covered are eliminated from a set of cubes, the cubes remaining when the generation procedure terminates are prime implicants. The set of ΔV_i vectors provides a means of separating the undesirable cubes from the prime implicants. These vectors were used in building the cubes during cube generation, and they may be employed to determine which cubes are contained in larger cubes. If a j-cube is a prime implicant, the vertices which complete the j $(j-1)$-cubes are covered by the prime implicant and may be found by subtracting in turn the ΔV_i's from the vertex that completes the j-cube, V_i^j. Theorem 4 reduces the calculations in determining these subcubes.

THEOREM 4. *If U^j is equal to U_c^j, then for all k less than j, each implicant completed by a member of U_c^k is covered by a j-cube.*

PROOF. Each smaller cube was used to build a larger cube since $U^j \equiv U_c^j$. Q. E. D.

A very special feature of this alogrithm is demonstrated when all 2^d vertices are members of the K-array. By Theorem 4 the d-cube is the only prime implicant that covers V^0. This sole cover of V^0 is called an *extremal* and *must* be a part of the minimum expression of the Boolean function.

Assume V^0 is covered by an extremal and consider all vertices other than V^0 that are also contained in that extremal. Each such vertex is covered by a cube which must be included in the minimum expression of the Boolean function and has exactly the same properties as the DON'T CARE vertices.

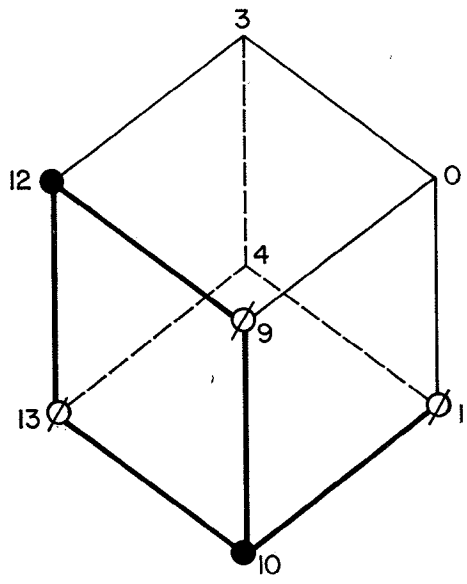
It should be noted that the procedure only finds prime implicants that cover at least one vertex, V^0, of the ON-array. This is not true in competitive procedures which do not differentiate between ON and DON'T CARE vertices during prime implicant generation and, in turn, generate many prime implicants that only cover vertices in the DON'T CARE-array and are therefore useless. The following set of sequential steps are listed to summarize prime implicant and extremal extraction.

Step. 6 If a d-cube cover or extremal exists, cast all 2^d vertices into the DON'T CARE-array; prime implicant extraction is completed. If not, the largest cubes are prime implicants; go to Step. 7.

Step 7. Use the ΔV_i vectors to determine if each implicant of the next smaller size is covered. Any implicant not covered becomes a prime implicant.

Step 8. Use Theorem 4 to determine whether prime implicant extraction is completed or whether to return to Step 7.

Example:



V^0 $U^1 = U_c^1$ U^2 $U_c^2 = \{12\}$

10/ 13 9 1 / 12 0 $U_{nc}^2 = \{0, 4\}$

4

U^3 is not generated since U_c^2 has less than 3 members.

$12 - 3 = 9$ } show that 1-cubes represented by vertices 9 and

$12 - (-1) = 13$ } 13 are covered by the 2-cube.

Therefore, the 2-cube and 1-cube represented by vertices

12 and 1 are prime implicants that cover vertex 10. Note, that an extremal will be found when $V^0 = 12$.

ON ARRAY = $\{10(A\bar{B}C),\ 12\ (AB\bar{C})\}$

DONT CARE ARRAY = $\{1(\bar{A}\bar{B}C),\ 9(A\bar{B}\bar{C}),\ 13(ABC)\}$

To develop a general procedure for determining the decimal representation of any cube, consider two adjacent vertices that form a 1-cube.

THEOREM 5. *The decimal representation of a 1-cube of two adjacent vertices is given by twice the larger vertex representation minus the smaller vertex representation.*

PROOF. If two vertices V_1 and V_2 are adjacent, they differ in one variable position, the larger containing a one and the smaller containing a zero or

$$V_1 = (d_1 \cdots 1 \cdots d_n)_3 \quad \text{and} \quad V_2 = (d_1 \cdots 0 \cdots d_n)_3 .$$

Then

$$2V_1 - V_2 = (d_1 \cdots 2 \cdots d_n)_3 .$$

From the trit assignments it follows that the operation $2V_1 - V_2$ effectively eliminates the variable that differs. Q.E.D.

Theorem 5 establishes the decimal representation for any 1-cube, C_i^1, which covers V^0. To find larger cubes which cover V^0, the vector approach may be used again. Define d vectors, ΔC_i, as:

$$C_i^1 = V^0 + \Delta C_i \tag{3}$$

These ΔC_i vectors may be described as the decimal numbers which, when added to V^0, eliminate the variable in the position of adjacency between V^0 and V_i^1.

A relationship exists between the ΔV_i's and the ΔC_i's.

THEOREM 6. *If* $\Delta V_i < 0$, *then* $\Delta C_i = -\Delta V_i$; *and if* $\Delta V_i > 0$, *then* $\Delta C_i = 2(\Delta V_i)$.

PROOF. If $\Delta V_i = V_i^1 - V^0 < 0$ or $V_i^1 < V^0$, then by Theorem 5,

$$C_i^1 = 2V^0 - V_i^1$$
$$\Delta C_i = C_i^1 - V^0 = (2V^0 - V_i^1) - V^0$$
$$= V^0 - V_i^1 = -\Delta V_i .$$

If $\Delta V_i = V_i^1 - V^0 > 0$ or $V_i^1 > V^0$, by Theorem 5,

$$C_i^1 = 2V_i^1 - V^0$$
$$\Delta C_i = C_i^1 - V^0 = (2V_i^1 - V^0) - V^0$$
$$= 2(V_i^1 - V^0) = 2(\Delta V_i). \qquad \text{Q.E.D.}$$

The effect of adding j ΔC_i's to V^0 is to eliminate j variables and form a j-cube since each ΔC_i eliminates a unique variable. The vertex V_i^j is found by adding j ΔV_i's to V^0 and represents a cube. The ΔV_i vectors define the j dimensions of the cube. If the set of respective ΔC_i vectors are added to V^0, the j literals are eliminated, producing the encoded cube. Computation of cubes may parallel the procedures for locating the vertices which complete those cubes. The foregoing is summarized by the following algorithm for finding the decimal representation of prime implicants.

Step 9. Use Theorem 5 to calculate the ΔC_i's from the ΔV_i's.

Step 10. Use eq. 3 to calculate all d members of the set of C_i^1's.

Step 11. With the ΔC_i's replacing the ΔV_i's, use the Generation Rules to calculate the decimal representation of each prime implicant.

Example (continued):

$$\Delta C_1 = 2 \cdot 3 = 6 \qquad C_1^1 = 2 \cdot 13 - 10 = 16$$
$$\Delta C_2 = -(-1) = 1 \qquad C_2^1 = 2 \cdot 10 - 9 = 11$$
$$\Delta C_3 = -(-9) = 9 \qquad C_3^1 = 2 \cdot 10 - 1 = 19$$

$$10/16 \cdot 11 \quad 19/17 \quad \begin{matrix} 20 \\ 25 \end{matrix}$$

Therefore, $19(201_3 = \bar{B}C)$ and $17(122_3 = A)$ are the two prime implicants that cover vertex 10. Steps 1–11 constitute a means of finding every prime implicant that covers a single vertex V^0 in the ON-array. To cover the Boolean function, every member of the ON-array that is not cast into the DON'T CARE-array by step 6 must be covered. Therefore steps 1–11 must be repeated until this is completed.

This algorithm was programmed in FORTRAN. The program was extremely efficient when applied to the class of Boolean functions that are covered predominantly by extremals and could be very efficient applied to the class of functions with many DON'T CARE vertices. In addition, the algorithm inherently requires significantly less memory since the method only analyzes the portion of the function containing V^0 to find the prime implications covering V^0. In a problem containing n variables, there are only $\binom{n}{i}$ possible i-cubes containing vertex V^0. In contrast, there are $\binom{n}{i} \cdot 2^{n-1}$ possible i-cubes within the n-cube and such procedures as Quine-McCluskey [6] may generate lists of this size in the process of finding the same prime implicants. As a last observation, the method is not as efficient using decimal arithmetic as it is capable of being using binary coding and set operations. Decimal arithmetic and FORTRAN were chosen for versatility and convenience in programming.

RECEIVED SEPTEMBER, 1966; REVISED SEPTEMBER, 1967

REFERENCES

1. URBANO, R. H., AND MUELLER, R. K. A topological method for the determination of the minimal forms of a Boolean function. *IRE TEC 5*, 3 (Sept. 1956), 126–132.
2. HARRIS, B. An algorithm for determining minimal representations of a logic function. *IRE TEC 6*, 2 (June 1957), 103–108.
3. PRATHER, R. Computational aids for determining the minimal form of a truth function. *J. ACM 7*, 4 (Oct. 1960), 299–310.
4. DULEY, JAMES R. A decimal algorithm for minimizing Boolean functions. M. S. Thesis, U. of Wisconsin, 1963.
5. ROTH, J. P. Algebraic topological methods for the synthesis of switching systems I, *Trans. Amer. Math. Soc. 88*, (July 1958), 301–326.
6. MILLER, R. E. *Switching Theory, Vol. 1, Combinational Circuits*. Wiley, New York, 1965.