JOURNAL OF ALGORITHMS 13, 546–563 (1992)

A Polynomial Algorithm for Minimal Interval Representation*

ANAT FISCHER

The Technion, Israel

ITZHAK GILBOA

Northwestern University, Evanston, Illinois 60208

AND

MOSHE SHPITALNI

The Technion, Israel and
University of California, Santa Barbara, California 93106

Received June 29, 1988; revised August 1991

The following problem arises in the context of object representation: given two endpoints of an interval in a Gray code table, find a Boolean function in DNF that represents this interval, with as few prime implicants as possible. This paper shows that there is a unique minimal representation and presents a polynomial algorithm that finds it. © 1992 Academic Press, Inc.

1. INTRODUCTION

A three-dimensional object has many possible representations inside a computer. Specifically, constructive solid geometry, boundary representation, and volumetric approximation techniques are commonly used to represent objects in solid geometric modeling (see [4]). Among the volumetric schemes are the octree [3] and the more recently developed switching-function representation [6] in which the object is modeled as occupying precisely those points for which the switching function value

*We are very grateful to three anonymous referees for many comments and references. We are especially indebted to one of them whose report was more than twice as long as the original paper and provided numerous stylistic and a few mathematical corrections.

0196-6774/92 $5.00

is 1. Both schemes are based upon cell decomposition and approximating objects to a desired resolution. Consequently, they both share many properties and yet each of them has its own advantages and drawbacks. One of the main drawbacks of the octree scheme is its exponential space complexity. By contrast, this paper is concerned with a polynomial-time algorithm to generate the representation of an object by a switching-function, where the space is modeled as a Gray-code table (as in [6]). The minimal representation of an object via a switching function entails the generation of a minimal set of prime implicants which covers the object. The generation of such a set is known to be NP-complete in general (see [1, 5]). However, it will be shown that for contiguous segments, which represent the practical cases of connected objects, the minimal set of prime implicants is unique and can be generated in polynomial time.

2. The Main Result

The following definitions are standard ones (see, for example, [2]) and are given for the sake of completeness and convenience of notation.

A *switching function* of n variables is a function f: $\{0,1\}^n \to \{0,1\}$. An *implicant* of n variables is a function P: $\{1,2,\ldots,n\} \to \{0,1,d\}$ (where d stands for "don't care"). To each implicant of n variables P we attach a switching function of n variables $\tilde{P}$ defined as follows: $\tilde{P}(x_1,\ldots,x_n) = 1$ iff for all $1 \le i \le n$ one of the three conditions holds: (i) $P(i) = d$; (ii) $P(i) = 1$ and $x_i = 1$; (iii) $P(i) = 0$ and $x_i = 0$. If $\tilde{P}(x_1,\ldots,x_n) = 1$, then P is said to *cover the point* $\mathbf{x} = (x_1,\ldots,x_n)$.

In the sequel we may refer to "an implicant of $x_1,\ldots,x_i$", which will be a function from $\{1,\ldots,i\}$ to $\{0,1,d\}$. However, it will be convenient to abuse notation and refer to such an implicant also as an implicant of any superset of $\{x_1,\ldots,x_i\}$, where the value of each of the other variables is assumed to be "d."

For each switching function f we consider the set $\{\mathbf{x} = (x_1,\ldots,x_n) | f(\mathbf{x}) = 1\}$. Since no confusion is likely to arise, we will denote this set by f as well.

An implicant P is an *implicant of the function* f if $\tilde{P} \subseteq f$. It is a *prime implicant* of f if there is no other implicant P' such that $\tilde{P} \subseteq \tilde{P}' \subseteq f$.

A *code table*, or simple a *table*, T, of size n is a permutation of all 2^n Boolean vectors of length n. Formally, T is a bijection from $\{k | 0 \le k \le 2^n - 1\}$ onto $\{\mathbf{x} | \mathbf{x}: \{1,\ldots,n\} \to \{0,1\}\}$. Thus, $T(k)$ is a vector for $0 \le k \le 2^n - 1$, and for all $1 \le j \le n$, $T(k)(j) \in \{0,1\}$. When no confusion is likely to result, we will allow ourselves to use $T(k)$ and k interchangeably.

A table T is said to have the *distance-1 property* if whenever $k - l \equiv 1 \pmod{2^n}$, $T(k)$ and $T(l)$ differ by one bit exactly.

The *Gray code table* of size n, GT_n, is defined inductively as

(1) For $n = 1$, $GT_1(0)(1) = 0$ and $GT_1(1)(1) = 1$.

(2) For $n > 1$, GT_n is obtained from GT_{n-1} as follows: write each row of GT_{n-1} twice, and fill out the nth bit by repeating the sequence 0110. Formally, for $0 \le k \le 2^{n-2} - 1$ and $j < n$,

Duplicate:
$GT_n(4k)(j) = GT_{n-1}(2k)(j)$
$GT_n(4k + 1)(j) = GT_{n-1}(2k)(j)$
$GT_n(4k + 2)(j) = GT_{n-1}(2k + 1)(j)$
$GT_n(4k + 3)(j) = GT_{n-1}(2k + 1)(j)$
and Pad:
$GT_n(4k)(n) = 0$
$GT_n(4k + 1)(n) = 1$
$GT_n(4k + 2)(n) = 1$
$GT_n(4k + 3)(n) = 0$.

For convenience, we will also refer to GT_0 as the table containing a single empty vector. It is both well known and easy to prove that Gray code tables have the distance-1 property.

A set of implicants $\{P_i\}_{i=1}^k$ *represents* a function f if $f = \sum_{i=1}^k \tilde{P}_i$. It forms a *prime implicant representation* of f if each P_i is a prime implicant of f. It is a *minimal prime implicant representation* if it is minimal w.r.t. (with respect to) the number k of prime implicants.

This definition of "minimality" is weaker than the usual ones (see, for instance, [2]). However, we will prove that if f is a contiguous interval in a Gray code table, then it has a *unique* minimal representation according to this definition. Hence, this representation is also minimal with respect to any stronger criterion.

We may now formally define the problem addressed in this paper:

Minimal interval representation. Given two integers, l and m, such that $0 \le l \le m$, find a minimal prime-implicant representation of the switching-function f defined by the interval $[l, m]$ in the Gray code table of size $n \equiv \lceil \lg(m + 1) \rceil$; that is, $f(\mathbf{x}) = 1$ iff $l \le GT_n^{-1}(\mathbf{x}) \le m$.

THEOREM. *There exists a polynomial-time algorithm which solves the problem of minimal interval representation.*

3. PROOF

We begin with a verbal description of the main idea, followed by an example. Only then do we turn to the formal description of the algorithm and the proofs of its correctness and time complexity.

3.1. *Description of the Algorithm*

The main idea of the algorithm is as follows: given the interval $[l, m]$, first consider the last variable, x_n. It may be the case that its value can be flipped in both $GT_n(l)$ and $GT_n(m)$ without leaving the interval $[l, m]$. In this case there is no need to have x_n involved in any prime implicant, the last bit may be ignored and we have to consider a new problem in GT_{n-1}.

Suppose that one of $\{GT_n(l), GT_n(m)\}$ (or both) cannot have its nth bit flipped without leaving the interval $[l, m]$. Such a point will be referred to as an "edge point." Suppose $GT_n(l)$ is such. In this case every prime implicant representation has to have at least one prime implicant with a specific value (different than "d") for x_n, in order to cover l but not the adjacent point, $l - 1$. It will turn out to be the case that there is a maximal such prime implicant (w.r.t. set inclusion) and it is obtained by checking which bits in $GT_n(l)$ (alternatively, in $GT_n(m)$) may be *separately* flipped without leaving $[l, m]$ and setting *all of them* to "d."

After writing down the prime implicants obtained by this procedure we may again ignore the last bit and continue as before. However, one problem remains: if x_n's value at $GT_n(l)$ was 1, say, and at $GT_n(m)$ it was 0, a prime implicant with $x_n = 1$ and one with $x_n = 0$ may jointly cover some points in GT_{n-1}. Hence, we have to check whether such prime implicants may be "reduced" to implicants involving only $x_1, \ldots, x_{n-1}$.

The algorithm continues in this way, with only edge points of the interval being checked for possible inclusion in the union of already-obtained prime implicants, since only edge points may trigger the introduction of new prime implicants to the representation. This algorithm is proven in Subsection 3.4 to be correct. Subsection 3.5 shows that it is of time complexity $O(n^3)$.

3.2. *An Example*

Suppose we are given $l = 3$, $m = 14$; hence $n = 4$. The table and the interval are presented in Fig. 1.

Here both l and m are edge points, hence they both require implicants involving x_4: if $l = 3$ is covered by an implicant P with $P(4) = d$, P will also cover the point $l - 1 = 2$; similarly, if $m = 14$ will be covered by such a P, P will also cover $m + 1 = 15$.

Let us start with the implicant P that will cover l. Obviously, $P(4) = 0$. Now we have to check for each of the bits $x_1 - x_3$ *separately* whether it can be flipped without leaving [3, 14]. Flipping bits 1, 2, and 3 will yield points 12, 4, and 0, respectively. Since $4, 12 \in [3, 14]$, we conclude that we must define $P(1) = P(2) = d$; $P(3) = 1$. That is, the implicant P is $x_3\bar{x}_4$ (where the upper bar stands for negation). Note that this implicant covers the points 3, 4, 11, and 12. While we specifically checked only the points

	x_1	x_2	x_3	x_4	
0	0	0	0	0	
1	0	0	0	1	
2	0	0	1	1	
3	0	0	1	0	ℓ = 3
4	0	1	1	0	
5	0	1	1	1	
6	0	1	0	1	
7	0	1	0	0	
8	1	1	0	0	
9	1	1	0	1	
10	1	1	1	1	
11	1	1	1	0	
12	1	0	1	0	
13	1	0	1	1	
14	1	0	0	1	m = 14
15	1	0	0	0	

FIGURE 1.

obtained by flipping one bit—4 and 12—the implicant also covers those obtained by flipping several bits—in this case, 11—but, fortunately, they end up lying inside the interval. (We will later prove that this is a rule rather than coincidence.)

Similarly, we compute the implicant Q that will cover $m = 14$. Flipping bits 1, 2, and 3 yields the points 1, 9, and 13, respectively. The last two are in $[3, 14]$. Hence, we define $Q(1) = 1$, $Q(2) = Q(3) = d$, and $Q(4) = 1$; that is, Q is the implicant x_1x_4. This implicant covers the points 9, 10, 13, and 14. The point 10 is obtained by flipping several (two) bits, and lies inside $[3, 14]$. The points already covered are shown in Fig. 2.

When reducing the table to size $i = 3$, we surely want to exclude the two edge points already covered from the updated interval. (For instance, covering the point 001 in GT_3 would also mean covering 0011 in GT_4.) Dropping the edge points, ignoring the fourth bit and eliminating redundant rows, we obtain the new interval in GT_3, presented in Fig. 3.

Note that both points 5 and 6 are already jointly covered by the implicants $x_3\bar{x}_4$ and x_1x_4. Yet, we do not drop these points. During the execution of the algorithm it is more accurate to think of the interval $[l, m]$ as the maximal range that *may* be covered by the prime implicants, not as the (precise) range that *has* to be covered. Note that if the original interval, $[l, m]$, is not the whole table (i.e., if $l > 0$ or $m < 2^n - 1$), every

	x_1	x_2	x_3	x_4	
0	0	0	0	0	
1	0	0	0	1	
2	0	0	1	1	
3*	0	0	1	0	$\ell = 3$
4*	0	1	1	0	
5	0	1	1	1	
6	0	1	0	1	
7	0	1	0	0	
8	1	1	0	0	
9*	1	1	0	1	
10*	1	1	1	1	
11*	1	1	1	0	
12*	1	0	1	0	
13*	1	0	1	1	
14*	1	0	0	1	$m = 14$
15	1	0	0	0	

$* = x_3\bar{x}_4 + x_1x_4$.

FIGURE 2.

	x_1	x_2	x_3	
0	0	0	0	
1	0	0	1	
2	0	1	1	$\ell = 2$
3	0	1	0	
4	1	1	0	
5	1	1	1	
6	1	0	1	$m = 6$
7	1	0	0	

FIGURE 3.

point in it will eventually be included in an edge point of the updated interval in GT_i for some $i \leq n$.

However, to guarantee the minimality of the representation obtained we have to consider only edge points that are not already covered. Consider Fig. 3 again. Here, $l = 2$ is not an edge point (flipping the third bit yields the point $3 \in [2, 6]$); however, $m = 6$ is an edge point (flipping its third bit

	x_1	x_2	
0	0	0	
1	0	1	$\ell = 1$
2	1	1	$m = 2$
3	1	0	

FIGURE 4.

yields $7 \notin [2, 6]$). Yet 6 is already covered by $x_3\bar{x}_4 + x_1x_4$, so we disregard it.

In order to know whether edge points in table GT_i are already covered, the algorithm will update a list of temporary implicants. Thus, $x_3\bar{x}_4$ and x_1x_4 will be introduced into the list of the first stage, say, PTEMP[4], and before analyzing GT_3 we will add to PTEMP[3] the implicant x_1x_3, as every point (in GT_3) covered by it is already covered by $(x_3\bar{x}_4 + x_1x_4)$.

Thus, in GT_3 there is no need for additional implicants. The edge point $m = 6$ may be ignored and we are left with the interval in GT_2 shown in Fig. 4.

There is only one temporary implicant in PTEMP[3], which is x_1x_3. Hence, all the implicants in PTEMP[3], and hence all the prime implicants gathered so far, $(x_3\bar{x}_4 + x_1x_4)$, do not cover any points in GT_2. We may therefore proceed as if this were the original table. Here, both $l = 1$ and $m = 2$ are edge points, and both induce the same prime implicant, x_2. This prime implicant is added to the list, and obviously we are left with an empty interval (this will be detected by the algorithm when, after the control variables are updated, we have $l > m$). Therefore, the algorithm terminates with the representation

$$x_3\bar{x}_4 + x_1x_4 + x_2.$$

Following the steps of the algorithm, it is relatively easy, yet insightful, to convince oneself that this representation is indeed minimal.

3.3. *A Formal Description of the Algorithm*

Data Structure. The following global variables will greatly simplify the exposition:

(1) n, l, m, i, j.

(2) *PI*, a set of implicants of n variables (those maximal implicants which have to be included in the representation).

(3) For each $1 \leq i \leq n$, PTEMP[i], a set of implicants of i variables (to be thought of as a function of the first i variables $x_1, x_2, \ldots, x_i$). (These represent the implicants in GT_i which are obtained by unions of the implicants in GT_{i+1}.)

DEFINITIONS. (1) A vector $x \in \{0,1\}^i$ $(i \leq n)$ will also be called a *point*.

(2) $GT_i(k)$ (or, simply, k) is *included* in the interval $[l, m]$ if $l \leq k \leq m$. It is an *endpoint* if $k = l$ or $k = m$; $l(m)$ will be also called the upper (lower) endpoint of $[l, m]$.

(3) A point k is an *edge point w.r.t. the interval* $[l, m]$ if k is an endpoint of $[l, m]$, and the point $\mathbf{x}$, defined by flipping its ith bit, that is,

$$\mathbf{x}(j) = GT_i(k)(j), \qquad j < i$$
$$\mathbf{x}(i) = 1 - GT_i(k)(i),$$

is not included in the interval, namely, $GT_i^{-1}(\mathbf{x})$ (which equals $k + 1$ or $k - 1$) is smaller than l or greater than m. We will also use the terms "upper edge point" (should l be an edge point) and "lower edge point" (for m).

(4) For i, k, l, and m, such that $0 \leq l \leq k \leq m \leq 2^i - 1$, define a set $A(k, i, l, m)$ to be

$\{1 \leq j \leq i | \mathbf{x}$ is in the $[l, m]$ interval (i.e., $GT_i^{-1}(x) \in [l, m]$), where $\mathbf{x}$ is obtained from $GT_i(k)$ by changing the value of its jth bit: $\mathbf{x}(s) = GT_i(k)(s)$, for $s \neq j$, and $\mathbf{x}(j) = 1 - GT_i(k)(j)\}$.

PROCEDURES.

(1) Procedure ADD(k). Add to the set PTEMP[i] the prime implicant (of i variables) P defined as follows:

$$P(j) = \begin{cases} d, & \text{if } j \in A(k, i, l, m) \\ GT_i(k)(j), & \text{otherwise} \end{cases}$$

for $1 \leq j \leq i$. Also, add to the set PI the corresponding prime implicant (of n variables) P' defined by

$$P'(j) = \begin{cases} P(j), & 1 \leq j \leq i, \\ d, & i < j \leq n. \end{cases}$$

for $1 \leq j \leq n$.

(2) Procedure AUGMENT − PTEMP(i). For every two implicants, P_1 and P_2, in PTEMP[i] with $P_1(i) = 1$ and $P_2(i) = 0$, perform the following: first check whether for each $j < i$ it is true that $P_1(j) = P_2(j)$, $P_1(j) = d$, or $P_2(j) = d$ (or possibly all of the above). If so, add to

PTEMP[$i-1$] an implicant (of $i-1$ variables) $\bar{P}$ defined by

$$\bar{P}(j) = P_1(j), \quad \text{if } P_1(j) = P_2(j) \text{ or } P_2(j) = d;$$
$$\bar{P}(j) = P_2(j), \quad \text{otherwise.}$$

For every implicant P in PTEMP[i] with $P(i) = d$, and which is not already covered by any (single) implicant in PTEMP[$i-1$], introduce into PTEMP[$i-1$] the corresponding implicant $\bar{P}$ of $(i-1)$ variables, that is, $\bar{P}$ such that $\bar{P}(j) = P(j)$ for $j < i$.

THE ALGORITHM.

Input: integers l and m such that $0 \le l \le m$.
Output: the set PI of prime implicants.
Initialization:
Compute $n = \lceil \lg(m+1) \rceil$.
If $l = 0$ and $m = 2^n - 1$, define the trivial implicant P by $P(j) = d$ for $1 \le j \le n$, let $PI = \{P\}$, and stop. Otherwise, continue.
Let $i = n$
Let PI and PTEMP[j] (for all $1 \le j \le n$) be empty.
WHILE $i \ge 1$ and $l \le m$ do:
If l is an edge point of the interval $[l, m]$, and if l is not covered by any of the elements of PTEMP[i], then perform ADD(l).
Repeat the previous step for m (instead of l).
Perform AUGMENT − PTEMP(i).
Update control variables:
Set: $i = i - 1$;
$l = \lfloor (l+1)/2 \rfloor$
$m = \lfloor (m-1)/2 \rfloor$.

3.4. *Proof of Correctness*

ADDITIONAL DEFINITIONS. To facilitate the discussion, we will define two functions. The first, REDUCE, reduces a table of a given size i to a table of size $i-1$. The second one, EXPAND, does the opposite. However, they are not entirely symmetric, since the reduction may be accomplished by "deleting" any chosen bit, while expansion is always performed by adding a leading bit to a given table.

FUNCTION REDUCE(T, i, j, s).

Input: T, a table of size i, for $i \ge 1$
j, an integer $1 \le j \le i$
s, a bit value: $s \in \{0, 1\}$.

Output: T', a table of size $i - 1$.
Computation: Let $(k_r)_{r=0}^{2^{i-1}-1}$ be the set of indices satisfying:
(i) $T(k_r)(j) = s$, for $r \leq 2^{i-1} - 1$
(ii) $k_{r+1} > k_r$, for $r < 2^{i-1} - 1$.
For $0 \leq r \leq 2^{i-1} - 1$, define
(i) $T'(r)(l) = T(k_r)(l)$, if $0 \leq l < j$
(ii) $T'(r)(l) = T(k_r)(l + 1)$, if $j \leq l < i$.

FUNCTION EXPAND(T, i, s).
Input: T, a table of size i, for $i \geq 0$
s, a bit value: $s \in \{0, 1\}$.
Output: T', a table of size $i + 1$.
Computation: For $0 \leq r \leq 2^i - 1$ define:
$T'(r)(1) = s$
$T'(r)(j) = T(r)(j - 1)$ for $2 \leq j \leq i + 1$.
For $2^i \leq r \leq 2^{i+1} - 1$, define:
$T'(r)(1) = 1 - s$
$T'(r)(j) = T(2^{i+1} - 1 - r)(j - 1)$
for $2 \leq j \leq i + 1$.
(The first half of T' is the concatenation of the leading bit s to T. The second half is a mirror image of T with leading bit $(1 - s)$.)

It is easy to see that both functions will, in fact, produce tables. (That is to say, if T is a permutation of all bit vectors, then T' will also be a permutation of all bit vectors of the corresponding size.)

We may now turn to the proof. Let us begin with some simple observations which will be given without proofs.

3.4.1. *Observation*. REDUCE(EXPAND(T, i, s), $i + 1, 1, s$) = T.

3.4.2. *Observation*. GT_{i+1} = EXPAND($GT_i, i, 0$).

3.4.3. *Observation*. GT_{i-1} = REDUCE($GT_i, i, i, 0$) = REDUCE ($GT_i, i, i, 1$).

(In fact, Observation 3.4.3 is a restatement of the definition of Gray code tables. Observation 3.4.2 may be used as an alternative definition.)

Let us define a *semi-Gray code table* (or a "semi-Gray code") of size n to be:

(i) the table containing the empty vector if $n = 0$, or

(ii) the table computed by EXPAND($T, n - 1, 0$) or EXPAND(T, $n - 1, 1$) for some semi-Gray code table T of size $n - 1$.

By Observation 3.4.2, Gray code tables may be obtained from the empty table by a sequence of EXPAND($\cdot, \cdot, 0$) operations. Semi-Gray codes are

those tables obtained by a similar process, when the leading bit (in the first half of the table) at each stage may be *either* 0 *or* 1. While there exists a unique Gray code table of size n, there are 2^n semi-Gray code tables of the same size. In fact, for each $0 \le k \le 2^n - 1$ there is a one-to-one correspondence between all 2^n semi-Gray codes and all 2^n possible n-bit vectors which may be the kth point in the table, i.e., for any choice of a vector $\mathbf{x}$ for any location $0 \le k \le 2^n - 1$ in the table, there exists a unique semi-Gray code T for which $T(k) = x$. Next we have:

3.4.4. LEMMA. *Let T be a semi-Gray code table of size $i \ge 1$. Then for any j, s such that $0 \le j \le i$ and $s \in \{0, 1\}$, REDUCE(t, i, j, s) is also a semi-Gray code table.*

Proof. The proof is by induction on i. For $i = 1$, it is obviously true. Assume then that $i > 1$. If $j = 1$, the definition of semi-Gray code tables is all that is required for a proof. Let us therefore consider the case $j > 1$.

Consider $T' = \text{REDUCE}(T, i, 1, T(1)(1))$ (that is, the upper half of T with its first bit deleted). T' is a semi-Gray code table by definition. Hence the induction assumption implies that $T'' \equiv \text{REDUCE}(T', i - 1, j - 1, s)$ is also a semi-Gray code table. It only remains to note that $\text{EXPAND}(T'', i - 2, T(1)(1)) = \text{REDUCE}(T, i, j, s)$. □

3.4.5. LEMMA. *Let T be a semi-Gray code table of size i. Assume that $j_1, j_2, \ldots, j_k$ satisfy $1 \le j_1 < j_2 < \cdots < j_k \le i$ and that $s_l \in \{0, 1\}$ for $1 \le l \le k$. Define $T_0 = T$ and $T_l = \text{REDUCE}(T_{l-1}, i - l + 1, j_l - l + 1, s_l)$ for $1 \le l \le k$. Then T_k is a semi-Gray code table.*

Proof. Follows from the previous lemma. □

3.4.6. LEMMA. *If T is a semi-Gray code table, then it has the distance -1 property.*

Proof. By induction using the definition of semi-Gray code tables. □

3.4.7. THEOREM. *Let $[l, m]$ be an interval in a Gray code table of size i. Let $\mathscr{P}$ denote the set of all implicants P that cover $l(m)$ and are included in $[l, m]$; that is,*

$$\mathscr{P} = \{P \mid \tilde{P}(l) = 1, \text{ and } l \le k \le m \text{ whenever } \tilde{P}(GT_i(k)) = 1\}$$
$$\left(\text{or } \mathscr{P} = \{P \mid \tilde{P}(m) = 1, \text{ and } l \le k \le m \text{ whenever } \tilde{P}(GT_i(k)) = 1\} \text{ for } m\right).$$

Then $\mathscr{P}$ has a maximal element w.r.t. inclusion; that is, there exists $Q \in \mathscr{P}$ such that for all $P \in \mathscr{P}$ we have $\tilde{Q}(GT_i(k)) = 1$ whenever $\tilde{P}(GT_i(k)) = 1$.

Proof. Suppose $[l, m]$ is given. We will only provide the proof for l, as it is symmetric for m. Define

$$Q(j) = \begin{cases} d, & \text{if } j \in A(l, i, l, m) \\ GT_i(l)(j), & \text{otherwise} \end{cases}$$

for $1 \le j \le i$.

Note that Q is the implicant defined by Add(l) in the algorithm.

Two claims complete the proof:

CLAIM 1. If $P \in \mathscr{P}$ then $\tilde{P} \subseteq \tilde{Q}$.

Proof. In view of the definition of $A(l, i, l, m)$, every x_j $(1 \le j \le i)$ whose value can be switched to "d" without covering points outside $[l, m]$ has indeed a "d" value in Q. □

CLAIM 2. *Q is included in $[l, m]$; that is, for all $0 \le k \le 2^i - 1$, $\tilde{Q}(GT_i(k)) = 1$ implies $l \le k \le m$.*

Proof. Assume the contrary. Hence, the set $K \equiv \{k | \tilde{Q}(GT_i(k)) = 1, k < l \text{ or } k > m\}$ is nonempty. Let k be the element of K that maximizes $(k - m) \bmod 2^i$, that is, the index of the point closest to l when distance is measured along the unidirectional path $m, m + 1, m + 2, \ldots, 2^i - 1, 0, 1, \ldots, l - 1, l$.

Let us now denote by $D(k, l)$ the set $\{1 \le j \le i | GT_i(k)(j) \ne GT_i(l)(j)\}$. By the definition of Q, $D(k, l) \subseteq A(l, i, l, m)$. We claim that $|D(k, l)| \ge 2$: the possibility $|D(k, l)| = 0$ is, of course, ruled out since $k \ne l$. However, $|D(k, l)| = 1$ is also impossible, since it contradicts the definition of $A(l, i, l, m)$. Thus, $|D(k, l)| \ge 2$ is established.

To focus on the different bits of $GT_i(k)$ and $GT_i(l)$, we will apply the REDUCE function to the table, and reduce all of the bits in which they are equal. Suppose, then, that those bit positions are numbered $j_1, j_2, \ldots, j_r$, where $j_1 < j_2 < \cdots < j_r$. Define $T_0 = GT_i$ and, for $1 \le s \le r$, let $T_s = \text{REDUCE}(T_{s-1}, i - s + 1, j_s - s + 1, GT_i(l)(j_s))$. Let us denote the elements of $D(k, l)$ by $d_1, d_2, \ldots, d_t$, where $d_1 < d_2 < \cdots < d_t$ (*Note.* $t + r = i$). We may now consider the points in T_r which correspond to $GT_i(l)$ and $GT_i(k)$: let $\mathbf{x}(s) = GT_i(l)(d_s)$ and $\mathbf{y}(s) = GT_i(k)(d_s)$, for $1 \le s \le t$, and define $l' = T_r^{-1}(\mathbf{x})$, $k' = T_r^{-1}(\mathbf{y})$. Note that by the above construction, $\mathbf{x}(s) \ne \mathbf{y}(s)$ for all $1 \le s \le t$.

We now claim that $k' \equiv l' - 1 \pmod{2^t}$. To prove this it suffices to note that had there been another vector in T_r between $\mathbf{x}$ and $\mathbf{y}$, k could not have maximized $(k - m) \pmod{2^i}$. More formally, if $\mathbf{z}$ satisfies $T_r^{-1}(\mathbf{y}) < T_r^{-1}(\mathbf{z}) < T_r^{-1}(\mathbf{x})$ or $T_r^{-1}(\mathbf{z}) < T_r^{-1}(\mathbf{x}) < T_r^{-1}(\mathbf{y})$ or $T_r^{-1}(\mathbf{x}) < T_r^{-1}(\mathbf{y}) < T_r^{-1}(\mathbf{z})$, one may define a bit-vector $\mathbf{w}$ of length i by $\mathbf{w}(d_s) = \mathbf{z}(s)$ for $1 \le s \le t$, and $\mathbf{w}(j) = GT_i(l)(j) = GT_i(k)(j)$ for $j \notin D(k, l)$, and then $GT_i^{-1}(\mathbf{w}) - m$ will exceed $k - m \pmod{2^i}$.

So $\mathbf{x}$ and $\mathbf{y}$ are adjacent points in T_r that differ in every bit position. But $t = |D(k, l)| \geq 2$ and, hence, T_r does not satisfy the distance-1 property. But this contradicts either Lemma 3.4.5 or 3.4.6. We therefore conclude that Claim 2 and the theorem are proved. □

This theorem implies that the algorithm computes a prime implicant of the interval $[l, m]$ in each stage. It is not difficult to see (as we will) that all of these prime implicants (computed at different stages) constitute a prime implicant representation of the original interval. However, we also need to show its minimality. To this end, we require several additional definitions.

For a subset $S \subseteq \{0, \ldots, 2^n - 1\}$ in a Gray code table and an integer $1 \leq i < n$, define the *reduction of S to* GT_i as $S|_i = \{\mathbf{x}: \{1, \ldots, i\} \to \{0, 1\} | .$ For every $\mathbf{y}: \{i + 1, \ldots, n\} \to \{0, 1\}$, $GT_n^{-1}(\mathbf{x} \cup \mathbf{y}) \in S\}$, where $\mathbf{x} \cup \mathbf{y}$ is the union of $\mathbf{x}$ and $\mathbf{y}$ (considered as sets), namely, the function on $\{1, \ldots, n\}$ defined by

$$\mathbf{x} \cup \mathbf{y}(j) = \begin{cases} \mathbf{x}(j), & 1 \leq j \leq i; \\ \mathbf{y}(j), & i < j \leq n. \end{cases}$$

Denote by I the input interval $[l, m]$ in GT_n.

3.4.8. *Observation.* After every update of the control variables in the algorithm, $[l, m]$ is $I|_i$, that is, the reduction of the original interval I to GT_i.

3.4.9. *Observation.* After every update of the control variables in the algorithm, the subsets of GT_i

$$P_i = \left(\bigcup_{P \in PI} \tilde{P} \right) \Big|_i$$

$$PT_i = \left(\bigcup_{P \in \text{PTEMP}[i]} \tilde{P} \right) \Big|_i$$

and

$$I_i = I|_i$$

satisfy

$$P_i = PT_i \subseteq I|_i.$$

This observation simply states that the sets PTEMP[i] represent exactly those points in GT_i which are already covered by prime implicants in PI, and that these do not "overflow" outside of the interval we are trying to represent.

For an implicant P, define the *order* of P to be the maximal index of a "relevant" variable, that is, the index $j \in \{0, \ldots, n\}$ for which: (i) $P(k) = d$ for all $k > j$ and (ii) $j = 0$ or $P(j) \neq d$.

3.4.10. LEMMA. *Suppose that P is a prime implicant of $I = [l, m]$ of order $j > 0$. Then P covers an edge point of $I|_j$ in GT_j.*

Proof. Otherwise setting $P(j) = d$ would provide a new implicant which is still included in $I|_j$ (in GT_j), hence in I (in GT_n), and P cannot be a prime implicant. □

We may now prove:

3.4.11. THEOREM. *For any interval $I = [l, m]$ in GT_n there is a unique minimal prime implicant representation given by the set PI computed by the algorithm.*

Note that in the provision of the theorem, "minimal" means "minimal with respect to the number of terms"; however, from the proof it will be obvious that it is enough to require minimality with respect to set inclusion. On the other hand, uniqueness will imply that the set PI is also minimal with respect to the number of factors in each term.

Proof. Existence of minimal representations is obvious. It therefore suffices to show that any minimal set of prime implicants $\mathscr{P}$ that represents I is equal to PI.

Given such a set $\mathscr{P}$, define for $0 \leq i \leq n$,

$$\mathscr{P}_i = \{P \in \mathscr{P} | P \text{ of order } i\}$$

and

$$PI_i = \{P \in PI | P \text{ of order } i\}.$$

We will prove that $\mathscr{P}_i = PI_i$ inductively. Let us begin with $i = n$. By Lemma 3.4.10, every element of both $\mathscr{P}_n$ and PI_n covers an edge point of I. On the other hand, every edge point has to be covered by a prime implicant in PI_n and by one in $\mathscr{P}_n$. Theorem 3.4.7 implies that these prime implicants are equal, and it follows that $PI_n = \mathscr{P}_n$. (Note that the above does not imply a one-to-one correspondence between edge points of I and $\mathscr{P}_n$ (or PI_n). It may well be the case that I has two edge points covered by the same prime implicant; but this prime implicant still has to be contained in both $\mathscr{P}_n$ and PI_n, which will not include any other prime implicants.)

Let us now assume that $\mathscr{P}_j = PI_j$ holds for $j > i$. As for the case $i = n$, it is true that

$$0 \leq |\mathscr{P}_i|, |PI_i| \leq 2,$$

and that each element of $\mathscr{P}_i$ and of PI_i has to cover an edge point of $I|_i$ in GT_i. Moreover, $PI_i \subseteq \mathscr{P}_i$ holds: given $Q \in PI_i$, Q covers an edge point not covered by any element of PTEMP[i]. By Observation 3.4.9, this point is not covered by $\bigcup_{\{P \in PI_j,\ j>i\}} \tilde{P}$ either. The induction hypothesis also implies that it is not covered by $\bigcup_{\{P \in \mathscr{P}_j,\ j>i\}} \tilde{P}$. Since an edge point cannot be covered by $P \in \mathscr{P}_j$ for $j < i$, there is $P \in \mathscr{P}_i$ covering the edge point, and by Theorem 3.4.7 it equals Q.

It remains to prove that the converse also holds, i.e., that $\mathscr{P}_i \subseteq PI_i$. Assume the contrary, namely, that $Q \in \mathscr{P}_i \setminus PI_i$. Let $\mathbf{x}$: $\{0, \ldots, i\} \to \{0, 1\}$ be an edge point of $I|_i$ covered by Q. If there were a $P \in PI_i$ covering this point, $P = Q$ would follow. Hence, no such P exists. In view of the algorithm, this is possible only if $\mathbf{x}$ is covered by

$$\bigcup_{\{P \in \text{PTEMP}[i]\}} \tilde{P}$$

and, by 3.4.9, also by

$$\bigcup_{\{P \in PI_j | j>i\}} \tilde{P}.$$

In particular, for every $\mathbf{y}$: $\{i+1, \ldots, n\} \to \{0, 1\}$ there exists $P_{\mathbf{y}} \in \bigcup_{j>i} PI_j$ covering $\mathbf{x} \cup \mathbf{y}$. (Again, this correspondence does not have to be 1–1.)

We wish to show that

$$\tilde{Q} \subseteq \bigcup_{\{\mathbf{y}:\ \{i+1, \ldots, n\} \to \{0,1\}\}} \tilde{P}_{\mathbf{y}} \subseteq \bigcup_{\{P \in PI_j,\ j>i\}} \tilde{P} = \bigcup_{\{P \in \mathscr{P}_j,\ j>i\}} \tilde{P},$$

which would imply that $\mathscr{P}$ is not minimal with respect to set inclusion, let alone with respect to the number of implicants.

Assume, then, that $\tilde{Q} \subseteq \bigcup_{\mathbf{y}} \tilde{P}_{\mathbf{y}}$ does not hold. Let $\mathbf{z}$ in GT_n be such that $\tilde{Q}(\mathbf{z}) = 1$ but $\tilde{P}_{\mathbf{y}}(\mathbf{z}) = 0$ for all $\mathbf{y}$. Choose $\mathbf{y}$ such that $\mathbf{y}(j) = \mathbf{z}(j)$ for $j > i$. $\tilde{P}_{\mathbf{y}}(\mathbf{z}) = 0$ can hold only if there is an integer $k \le i$ such that $P_{\mathbf{y}}(k) \neq d$ and $P_{\mathbf{y}}(k) \neq \mathbf{z}(k)$.

Recall that $\mathbf{x}$: $\{0, \ldots, i\} \to \{0, 1\}$ is the edge point of $I|_i$ covered by Q. W.l.o.g. assume that $\mathbf{x} = GT_j(l)$, that is, that $\mathbf{x}$ is an upper edge point. Hence, by the definition of $\tilde{P}_{\mathbf{y}}$,

$$\tilde{Q}(\mathbf{x} \cup \mathbf{y}) = 1 = \tilde{P}_{\mathbf{y}}(\mathbf{x} \cup \mathbf{y}).$$

Knowing that $P_{\mathbf{y}}(k) \neq d$, we obtain $P_{\mathbf{y}}(k) = \mathbf{x}(k)$ $(= (\mathbf{x} \cup \mathbf{y})(k))$. However, $\tilde{Q}(\mathbf{z}) = 1$ and $Q(\mathbf{x} \cup \mathbf{y}) = 1$ while $\mathbf{z}(k) \neq (\mathbf{x} \cup \mathbf{y})(k)$; hence $Q(k) = d$. This implies that $k \in A(l, i, l, m)$, namely, that changing the kth bit of $\mathbf{x}$ generates a point $\mathbf{x}'$ in the interval $I|_i$. But this further implies that

changing the kth bit of $\mathbf{x} \cup \mathbf{y}$ generates a point $\mathbf{x}' \cup \mathbf{y}$ within the original interval I. By the construction of prime implicants, $P_{\mathbf{y}}(k) = d$ must hold, a contradiction. This completes the proof of Theorem 3.4.11. □

3.5. *Proof of Time Complexity*

We first note that n is linear in the input size ($\lg m$), hence it suffices to show that the algorithm is polynomial in n. We show that the algorithm is of time complexity $O(n^3)$. Observe the following:

(1) Computing GT_i and GT_i^{-1} (i.e., computing a point in a Gray code table given its number, or the other way around), requires $O(i)$ operations.

(2) Computing $A(k, i, l, m)$ requires $O(i^2)$ operations.

We now show that at most one implicant generated by elements of PTEMP[i] is added to PTEMP[$i - 1$], which also means that the size of PTEMP[i] cannot exceed 3. This will imply that the construction of all PTEMP[i], $1 \leq i \leq n$, throughout the algorithm does not require more than $O(n^2)$ operations, and the $O(n^3)$ stated bound follows.

In the following, "stage j" will refer to the point in the algorithm's execution following the "control variables update" at which i is set to j ("stage n" will stand for the initial stage). Let l_j and m_j denote the values of l and m at state j.

3.5.1. LEMMA. *Suppose that at stages j' and j (with $j' < j$), $l_{j'}$ and l_j are upper edge points. Then*

$$GT_j(l_j)(k) = GT_{j'}(l_{j'})(k) \qquad \text{for } k < j'.$$

Furthermore, let P be the maximal prime implicant of $(x_1, \ldots, x_j)$ covering l_j, and let P' be the corresponding one for $l_{j'}$. Then for $k < j'$, if $P(k) = d$, then $P'(k) = d$ and, moreover, $P'(k) \neq d$ implies $P(k) = P'(k)$.

Proof. Note that the "moreover" part is a direct implication of the other two facts stated in the lemma. (Namely, if $P'(k) \neq d$, then $P(k) \neq d$ and, by equality of the Gray code table entry, $P(k) = P'(k)$ follows.)

It suffices to prove the lemma under the assumption that there are no upper edge points in any stage s, $j' < s < j$. (For, if there are such, one may proceed inductively.) Consider the point $GT_{j'}(l_{j'})$. It corresponds to some interval in GT_j. Let r be the uppermost point in this interval. Note that $r = l_j + 1$.

Since $l_{j'}$ is an upper edge point, flipping its j'th bit would result in a point outside $[l_{j'}, m_{j'}]$ in $GT_{j'}$. This means that

$$GT_j(l_j)(j') \neq GT_j(r)(j').$$

By the distance-1 property,

$$GT_j(l_j)(k) = GT_j(r)(k) \qquad \text{for } k \neq j',$$

and, in particular, for $k < j'$ we obtain

$$GT_j(l_j)(k) = GT_{j'}(l_{j'})(k).$$

Next consider the prime implicants P and P'. By the fact mentioned above and the symmetry of the Gray code table, it is obvious that the index r of the point (in GT_j) generated from l_j by flipping its kth bit ($k < j'$) is larger than any index s corresponding to a point in (the block induced by) $GT_{j'}(l_{j'})$. If r is in $[l_j, m_j]$, so is any such s, which means that when the kth bit is flipped in $l_{j'}$, in the table $GT_{j'}$, it does not generate a point outside of $[l_{j'}, m_{j'}]$. Hence, $P(k) = d$ implies $P'(k) = d$. □

A corresponding lemma can be proven and will be assumed, for lower edge points.

3.5.2. LEMMA. *The number of prime implicants generated from elements of* PTEMP[i] *and added to* PTEMP[$i - 1$] *is at most* 1.

Proof. The proof is by induction starting with $i = n$. For this case the claim is trivially correct. Consider some $i < n$, and assume correctness for $j > i$. Let us distinguish between two cases:

(1) $|\text{PTEMP}[i]| \leq 2$, in which case the lemma follows from the construction of PTEMP[$i - 1$].

(2) $|\text{PTEMP}[i]| = 3$.

First suppose that one of the elements of PTEMP[i], say, P, satisfies $P(i) = d$ (which can only occur if it was generated in previous stages). Then one of the two conditions must hold: (i) the other two implicants, say, P_1 and P_2, satisfy $P_1(i) = P_2(i)$, in which case only P is added to PTEMP[$i - 1$], or (ii) $P_1(i) \neq P_2(i)$ (and both differ from d by the construction of PTEMP[i] from edge points). In this case, P_1 and P_2 generate a prime implicant, $\bar{P}$, which is added to PTEMP[$i - 1$], but by Lemma 3.5.1, P will not be added to PTEMP[$i - 1$] as it is covered by $\bar{P}$.

Next assume that none of the three implicants has a "d" as its last entry. W.l.o.g., suppose $\{P_1, P_2, P_3\} = \text{PTEMP}[i]$ and $P_1(i) = P_2(i) = 1$, $P_3(i) = 0$. If P_1 was added to PTEMP[i] by AUGMENT-PTEMP($i + 1$), then a similar argument applies—the remaining two implicants P_2 and P_3 correspond to edge points and they generate an implicant $\bar{P}$ which covers P_1. Of course, the same holds if P_2 was added from PTEMP[$i + 1$]. So we are left with the case where P_1 and P_2 cover edge points, and P_3 was added to PTEMP[i] by AUGMENT-PTEMP($i + 1$).

But in this case, if a prime implicant, $\bar{P}_1$, is generated by AUGMENT-PTEMP for the pair P_1 and P_3, and $\bar{P}_2$ is generated by it for the pair P_2 and P_3, $\bar{P}_1$ and $\bar{P}_2$ have to be equal: for every $j < i$, if $P_3(j) = d$ then P_3 was generated by some lower and upper edge points with prime implicants P_4 and P_5 satisfying $P_4(j) = d$ and $P_5(j) = d$. Lemma 3.5.1 implies that $P_1(j) = d$ and $P_2(j) = d$ hold as well, whence $\bar{P}_1(j) = d$ and $\bar{P}_2(j) = d$.

If, however, $P_3(j) \neq d$, then $\bar{P}_1(j) = P_3(j) = \bar{P}_2(j)$, which completes the proof of Lemma 3.5.2. □

We conclude that the algorithm is of time complexity $O(n^3)$. Note that the only step which cannot be performed in $O(n^2)$ operations is the computation of $A(k, i, l, m)$.

Remark. The facts that n can be computed (rather than be given as input) and that this is crucial for n to be linear in the input size were pointed out to us by one of the referees.

REFERENCES

1. J. F. GIMPEL, A method of producing a Boolean function having an arbitrarily prescribed prime implicant table, *IEEE Trans. Electron. Comput.* **14** (1965), 485–488.
2. Z. KOHAVI, "Switching and Finite Automata Theory," McGraw-Hill, New York, 1970.
3. D. J. MEAGER, "The Octree Encoding for Efficient Solid Modeling," Ph.D. thesis, Graduate Faculty of Rensselaer Politechnic Institute, Troy, New York, submitted, 1982.
4. M. E. MORTENSON, "Geometric Modeling," Wiley, New York, 1985.
5. J. O'ROURKE, Polygon decomposition and switching function minimization, *Comput. Graphics Image Process.* **18** (1982), 382–391.
6. M. SHPITALNI, Switching functions and solid geometrical modeler, *Proc. IEEE* **72**, No. 1 (1984), 136–137.