TR-85-00456

TR-85-006

可程式邏輯陣的邏輯簡化

NSC74-0201-E001-01

郭譽申

周文光

鄒美蘭

中華民國七十四年十二月

CUP: a Multiple-Valued Logic Minimizer

Y.S. Kuo

Institute of Information Science
Academia Sinica

1. Introduction

Multiple-valued logic minimization has gained more and more attention due to its application to the design of programmable logic array (PLA). In principle, a PLA can implement any multiple-output switching function. Such a switching function can be treated as a Boolean function with multiple-valued inputs; the outputs of the switching function are treated as one multiple-valued input of the Boolean function. Moreover, a Boolean function with multiple-valued inputs is very useful in designing PLAs with decoders. A minimized PLA with t-bit decoders corresponds to a minimum sum-of-products expression for a Boolean function with 2^t-valued inputs. Other applications of multiple-valued logic minimization include the input-encoding problem, the optimal state-assignment problem and the optimal assignment of microcodes.

Many systems for two-level (AND-OR) logic minimization have been developed. But only a small fraction of these systems can handle multiple-valued inputs. Two well-known multiple-valued logic minimizers are MINI and ESPRESSO-MV. In order to handle many input variables in moderate computation time, both MINI and ESPRESSO-MV adopt heuristic algorithms and do not generate all prime implicants. They have similar overall structures and include major procedures such as COMPLEMENT (computing the off set of a function), EXPAND (expanding implicants into desirable prime implicants), ESSENTIAL (detecting which of the prime implicants are essential primes), IRREDUNDANT (extracting from a prime cover a minimal, irredundant cover) and REDUCE (reducing an implicant into the smallest implicant which together with the remaining implicants, still covers the original function).

In this paper, we desribe our effort in designing a new logic minimizer CUP (CUbe Processor) for a Boolean function with multiple-valued inputs. This system has similar major procedures as ESPRESSO-MV, but new algorithms have been developed. Feathers of CUP are as follows:

1. Essential primes are identified as early as possible.

ESPRESSO II and ESPRESSO-MV identifies essential primes after the EXPAND procedure has applied once and discovered a prime cover of the function. However, CUP can identify essential primes before a prime cover is discovered. Since essential primes must be in the final minimum sum-of-products expression for the Boolean function, early detection of essential primes can result in a problem which is smaller and usually easier than the original one.

2. A necessary and sufficient condition is derived for detecting essential primes.

Necessary and sufficient conditions for detecting essential primes for switching functions with binary inputs have been known for some time. As for multiple-valued switching functions, Sasao gave a sufficient condition for detecting essential primes recently. One step further, we shall prove a necessary and sufficient condition for essential prime detection in the multiple-valued case. The condition is computationally easy to check.

3. The expansion of implicants can be done in small and reasonable regions.

In MINI and ESPRESSO-MV, the expansion of an implicant is done within its overexpanded cube. Sometimes the overexpanded cube can cover a large region and contain many implicants of the function. This complicates the task of expansion. In CUP we adopt an approach which can reduce this region drastically. Note that implicants that can possibly be covered by an expansion of the current implicant are still within the small region and thus under consideration.

4. The EXPAND procedure expands implicants based upon a graph called the covering graph.

The covering graph can indicate whether two implicants can possibly be covered by a prime implicant of the function. Thus a clique (complete subgraph) of the graph corresponds to a set of implicants which have potential to be covered by a prime implicant. The EXPAND procedure identifies maximal cliques of the covering graph in order to generate prime implicants that can cover as many implicants of the function as possible.

5. A simple and fast algorithm is designed for extracting a minimal, irredundant cover.

ESPRESSO uses a sophisticated algorithm for the procedure IRREDUNDANT. But experiments have shown that this procedure does not have much effect on the final cover size since the REDUCE procedure can also generate an irredundant cover. Thus a simple and fast algorithm for IRREDUNDANT is developed for CUP.

6. Properties of unate functions are exploited to speed up tautology checking and the REDUCE procedure.

Tautology checking is a basic procedure used extensively in procedures ESSENTIAL and IRREDUNDANT. ESPRESSO-MV detects the unateness of a cover in order to speed up the checking of tautology as well as the REDUCE procedure. We shall present a more general notion of unateness and show that the same results as those for ESPRESSO-MV still hold. Thus tautology checking and REDUCE can be further improved in speed.

In Section 2 we introduce the notation for multiple-valued logic minimization. We also describe the overall structure of our minimization system. In Section 3 we describe the theory and an algorithm for detecting essential primes. Then in Section 4 we describe the graph-based EXPAND procedure. A new algorithm for the IRREDUNDANT procedure is presented in Section 5. In Section 6 the unateness of a function is exploited for tautology checking and the REDUCE procedure. Finally we try to draw some conclusions in the last section.

2. Problem Definition and Outline of Algorithm

Let us define some concepts in multiple-valued logic. A Boolean function with multiple-valued inputs is a mapping

$$f: P_1 \times P_2 \times \ldots \times P_n \rightarrow \{0,1\}$$

where $P_i = \{0,1,\ldots,p_i-1\}$. A generalized Boolean function with multiple-valued inputs is a mapping

$$F: P_1 \times P_2 \times \ldots \times P_n \rightarrow \{0,1,d\}.$$

Each element of $P_1 \times P_2 \times \ldots \times P_n$ is called a minterm. The on-set,off-set and dc-set of a (generalized) Boolean function F are the sets of minterms that have function values 1,0 and d, respectively, and will be denoted by ON(F), OFF(F) and DC(F). Note that a Boolean function can be characterized by its on-set and vice versa.

A cube is a Cartesian product $S_1 \times S_2 \times \ldots \times S_n$ where $S_i \subseteq P_i$ for all i = 1,2,...,n. The Boolean function associated with a cube is a product term. A cube t is an implicant of a (generalized) Boolean function F if $t \cap OFF(F) = \phi$. t is called a prime implicant if t is an implicant of F and there is no implicant t' of F with $t \subseteq t'$ and $t' \neq t$.

Let C be a collection of sets of minterms. Then

$$\cup C = \cup c$$

denotes the set of minterms covered by sets in C. A set C of implicants of function F is called a cover for F if $ON(F) \subseteq \cup C$. If every implicant of C is a prime implicant, then C is called a prime cover. Note that a cover for function F corresponds to a sum-of-products expression for F. Thus the (two-level) logic minimization problem is to find a prime cover for a generalized Boolean function of minimum size.

The CUP minimization system has the same basic structure as MINI and ESPRESSO. Let us outline its algorithm as follows:

Input: The on-set and dc-set of a generalized Boolean function F expressed as covers.

Output: A minimal, irredundant cover f for function F.

Algorithm CUP:

```
OFF = COMPLEMENT (ON U DC)
f = ON
Loop until no improvement on the cover size of f
    OVEREXPAND
    If first pass then ESSENTIAL
    EXPAND
    IRREDUNDANT
    REDUCE
Endloop
```

All procedures in the algorithm except OVEREXPAND have been described in Section 1. The OVEREXPAND procedure computes the overexpanded cube t^* for every implicant t in f. Note that the overexpanded cube t^* of t is the smallest cube containing all prime implicants of F which contain t, and can be computed easily if the off-set of F is known. The OVEREXPAND procedure also checks whether

$t^* \cap \text{OFF} = \phi$. If $t^* \cap \text{OFF} = \phi$, then t^* is a prime implicant. Actually t^* is the only prime implicant containing t. Thus we can replace t by t^* in f and delete those cubes in f which are covered by t^*.

The OVEREXPAND procedure can be considered as a preprocess for the EXPAND procedure. It sorts out those cubes that can be easily expanded. In the next section we shall show that essential primes for a Boolean function are among the overexpanded cubes.

3. Detection of Essential Primes

Given a generalized Boolean function F. A prime implicant p of F is an essential prime if p contains a minterm x in the on-set of F which is not contained in any other prime implicant. The minterm x is referred to as an essential point.

Any prime cover for a Boolean function contains all essential primes. Thus in most minimization systems, the detection of essential primes are performed after a prime cover has been discovered. But this can lead to pitfalls. As an example, the Karnaugh map of a Boolean function is shown in Fig. 1. Assume that the initial cover for the function consists of the minterms. If the minterm $\overline{A}B\overline{C}D$ is expanded first, it is likely that it will be expanded into $\overline{A}\overline{C}D$. Then when $\overline{A}BCD$ or ABCD is expanded, the product term BCD can be generated. Eventually we have the prime cover $\{\overline{A}\overline{C}D, BCD, \overline{A}\overline{B}\overline{C}, ABC\}$. Even though we can find out that $\overline{A}\overline{B}\overline{C}$ and ABC are essential primes, we still have a prime cover of size 4 while the minimum cover size is 3. Note that in this example neither IRREDUNDANT nor REDUCE can help reduce the cover size. However, if we can detect the essential primes $\overline{A}\overline{B}\overline{C}$ and ABC before the expansion, then those cubes covered by the essential primes can be excluded from the current cover. Consequently we can obtain the minimum prime cover $\{\overline{A}\overline{B}\overline{C}, ABC, \overline{A}BD\}$.

We can detect essential primes before the expansion process since every essential prime is an overexpanded cube.

Theorem 1 Let $C = \{t_1, t_2, \ldots, t_k\}$ be a cover for function F. If p is an essential prime of F, then

$p = t_j^*$ for some $j = 1,2,\ldots,k$.

Proof: Let x be an essential point contained in p. Since x is in the on-set of F, $x \in t_j$ for some j, $j = 1,2,\ldots k$. Since p is an essential prime, $t_j \sqsubseteq p$ and p is the only prime implicant containing t_j. Thus $p = t_j^*$.

If t_j^* is an essential prime, then $t_j^* \sqcap \text{OFF} = \emptyset$. The following result provides a fast check for essential primes.

Corollary 1 If $t_j^* \sqcap$ OFF $= \emptyset$ and t_j is a minterm in the on-set of F, then t_j^* is an essential prime of F.

Proof: Since $t_j^* \sqcap$ OFF $= \emptyset$, t_j^* is the only prime implicant containing t_j. This corollary then follows from the definition of essential primes.

To detect essential primes in general, we need the following operations.

Let t_1 and t_2 be cubes where

$$t_1 = S_1 \times S_2 \times \ldots \times S_n \text{ and } t_2 = R_1 \times R_2 \times \ldots \times R_n.$$

The sharp operation is

$$t_1 \# t_2 = \bigcup_{\substack{i=1,n \\ S_i \cup R_i \neq R_i}} S_1 \times \ldots \times S_{i-1} \times (S_i - R_i) \times S_{i+1} \times \ldots \times S_n$$

The asymmetric concensus of t_1 and t_2 is

$$\text{acons}(t_1,t_2) = \bigcup_{\substack{i=1,n \\ S_i \cup R_i \neq R_i}} (S_1 \sqcap R_1) \times (S_2 \sqcap R_2) \times \ldots \times (S_i \cup R_i) \times \ldots \times (S_n \sqcap R_n)$$

Note that the asymmetric concensus differs from the ordinary concensus operations (there are two commonly accepted definitions for concensus) only when $t_1 \sqcap t_2 \neq \emptyset$.

Lemma 1 Let C be a cover for function F, and let p be a prime implicant of F in C. Assume that

$$p \cap \cup(C \cup DC - \{p\}) = \emptyset .$$

Then p is an essential prime iff

$$p \cap ON \not\subseteq \cup \{ \text{acons}(h,p) \mid h \in C \cup DC - \{p\}\} .$$

Proof: Let $H = \cup \{ \text{acons}(h,p) \mid h \in C \cup DC - \{p\}\}$.

($\Longrightarrow$) Assume that p is essential.

Let x be an essential point in p. Suppose

$$x \in \text{acons}(h,p) \text{ for some } h \in C \cup DC - \{p\} .$$

Let $p = R_1 \times R_2 \times \ldots \times R_n$ and $h = S_1 \times S_2 \times \ldots \times S_n$. Since $p \cap \cup(C \cup DC - \{p\}) = \emptyset$, we have $p \cap h = \emptyset$ and

$$\text{acons}(h,p) = (S_1 \cap R_1) \times \ldots \times (S_i \cup R_i) \times \ldots \times (S_n \cap R_n)$$

Where $S_i \cap R_i = \emptyset$ and $S_j \cap R_j \neq \emptyset$ for $j \neq i$.

Let $x = (x_1, x_2, \ldots, x_n)$. Then

$$x_j \in S_j \cap R_j \text{ for all } j \neq i,\ j = 1,2,\ldots,n.$$

Consider a minterm $x' = (x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots x_n)$ where $y \in S_i$. Then $x' \in \text{acons}(h,p)$ but $x' \notin p$. We have shown

$$x \in \text{acons}(h,p) \not\subseteq p.$$

This contradicts the assumption that p is essential. Thus x can not be in any acons(h,p), $h \in C \cup DC - \{p\}$. In other words, $x \notin H$. Since $x \in p \cap ON$, $p \cap ON \not\subseteq H$.

($\Longleftarrow$) Assume that p is not an essential prime. Then for any minterm $x \in p \cap ON$, there exists an implicant t of F such that $x \in t$ and $t \not\subseteq p$. Let

$p = R_1 \times R_2 \times \ldots \times R_n$ and $t = S_1 \times S_2 \times \ldots \times S_n$. Then $S_i \not\subseteq R_i$ for some i. Let $x = (x_1, x_2, \ldots, x_n)$. Since $x \in p \cap t$, $x_j \in R_j \cap S_j$ for all $j = 1,2,\ldots,n$. Consider a minterm $x' =$

$(x_1,\ldots,x_{i-1},y,x_{i+1},\ldots,x_n)$ where $y \in S_i - R_i$.

Then $x' \in t$ and $x' \notin p$.

' p. Since t is an implicant of F and $x' \in t$, there exists a cube $h \in C \cup DC - \{p\}$ containing x'.

Let $h = T_1 \times T_2 \times \ldots \times T_n$. Since $p \cap \cup(C \cup DC - \{p\}) = \emptyset$, $p \cap h = \emptyset$. But $R_j \cap T_j \neq \emptyset$ for all $j \neq i$ because $x \in p$ and $x' \in h$. thus

$$\text{acons}(h,p) = (T_1 \cap R_1) \times \ldots \times (T_i \cup R_i) \times \ldots \times (T_n \cap R_n).$$

Since $x_j \in T_j \cap R_j$ for all $j \neq i$ and $x_i \in R_i$, $x \in \text{acons}(h,p)$.

Since x is an arbitrary minterm in $p \cap ON$, we have $p \cap ON \subseteq H$. This completes the proof.

Theorem 2 Let C be a cover for function F, and let p be a prime implicant of F in C. Then p is an essential prime iff

$$p \not\subseteq \cup\{\text{acons}(h,p) \mid h \in C \cup DC - \{p\}\} \cup DC.$$

Proof: We can construct a new cover C' and a new don't care set DC' such that they cover the same sets of minterms as C and DC, respectively, but

$$p \cap \cup(C' \cup DC' - \{p\}) = \emptyset .$$

This construction is done by taking a sharp operation on every cube t in $C \cup DC - \{p\}$ with $t \cap p \neq \emptyset$. Formally, $p \in C'$; if $t \in C - \{p\}$ and $t \cap p = \emptyset$, then $t \in C'$; if $t \in C - \{p\}$ but $t \cap p \neq \emptyset$, then $t \# p \subseteq C'$. DC' can be constructed from DC similarly. It should be noted that whether p is an essential prime is determined by the on, off and don't-care sets (of minterms), not by the covers for these sets. Therefore we can apply Lemma 1 to C' and DC', so p is an essential prime iff

$$p \cap ON \not\subseteq \cup\{\text{acons}(h,p) \mid h \in C' \cup DC' - \{p\}\}.$$

Let $H = \cup \{acons(h,p) \mid h \in C \cup DC - \{p\}\}$ and

$H' = \cup \{acons(h,p) \mid h \in C' \cup DC' - \{p\}\}$.

We shall show that H and H' cover the same set of minterms. Consider an arbitrary $h \in C \cup DC - \{p\}$ with $h \cap p \neq \emptyset$.

Let $p = R_1 \times R_2 \times \ldots \times R_n$ and $h = S_1 \times S_2 \times \ldots \times S_n$. Then

$$h \# p = \bigcup_{\substack{i=1,n \\ S_i - R_i \neq \emptyset}} h_i$$

where $h_i = S_1 \times \ldots \times S_{i-1} \times (S_i - R_i) \times S_{i+1} \times \ldots \times S_n \in C' \cup D' - \{p\}$.

Let us compute the asymmetric concensus.

$$acons(h,p) = \bigcup_{\substack{i=1,n \\ S_i \cup R_i \neq R_i}} (S_1 \cap R_1) \times \ldots \times (S_{i-1} \cap R_{i-1}) \times (S_i \cup R_i) \times (S_{i+1} \cap R_{i+1}) \times \ldots \times (S_n \cap R_n)$$

Since $h_i \cap p = \emptyset$ and $S_j \cap R_j \neq \emptyset$ for all j.

$$\begin{aligned} acons(h_i,p) &= (S_1 \cap R_1) \times \ldots \times (S_{i-1} \cap R_{i-1}) \times ((S_i - R_i) \cup R_i) \times \\ &\quad (S_{i+1} \cap R_{i+1}) \times \ldots \times (S_n \cap R_n) \\ &= (S_1 \cap R_1) \times \ldots \times (S_i \cup R_i) \times \ldots \times (S_n \cap R_n) \end{aligned}$$

Thus $acons(h,p) = \bigcup_{\substack{i=1,n \\ S_i \cup R_i \neq R_i}} acons(h_i,p)$.

WE have shown $H' = H$. Thus p is essential iff

$$p \cap ON \not\subseteq H$$

or equivalently, $p \not\subseteq H \cup DC$.

In Theorem 2 we have to check whether $p \subseteq H \cup DC$ or not. It has been shown that this can be done with a tautology checking algorithm.

4. The EXPAND Procedure

Let F be a generalized Boolean function, and let C be a cover for F. A prime cover E for F is called an expansion of C if for any implicant t in C, there is a prime implicant t' in E containing t. t' is called an expansion of t. The EXPAND procedure aims to find an expansion of a cover which is minimum in size. Even though the EXPAND procedure can have other secondary goals such as to expand implicants to as large as possible and to expand implicants such that their expansions overlap each other as much as possible, we will focus on the cover size only.

In MINI and ESPRESSO the EXPAND procedure expands cubes one at a time until all cubes have been either expanded into primes or covered by expansions of other cubes. In this expansion process each cube t is expanded within its overexpanded cube t^* and the algorithm aims to cover as many cubes in t^* as possible. We have recognized some weakness in this approach. First, the overexpanded cube t^* can be very large in size and cover a large number of cubes even though many of these cubes, actually, can not be covered by any expansion of t. Due to the large number of cubes involved, the expansion of a cube can not be done effectively. Another weakness in this approach is that the order for expanding cubes can influence the prime cover generated, but MINI and ESPRESSO adopt heuristic orderings which are rather arbitrary. The EXPAND procedure of CUP has been designed to make improvement on these two aspects. Before the EXPAND procedure is explained, we would like to remark that the existence of the preprocess OVEREXPAND has relieved the ordering problem to some extent according to the following lemma. Lemma

2 Let C be a cover for function F. If t^* is an overexpanded cube of t, $t \in C$, with $t^* \sqcap OFF = \emptyset$,then t^* must be in any expansion of C.
Proof: This is due to the fact that t^* is the only prime implicant containing t and any expansion of C must contain a prime implicant containing t.

In CUP the EXPAND procedure is based on a graph called the covering graph which indicate whether two cubes in cover C can possibly be covered by a prime implicant of function F. In this graph each vertex corresponds to a cube in C. Two vertices v_1 and v_2 are connected by an edge iff their corresponding cubes t_1 and t_2 satisfy the following condition

$$SCC(t_1,t_2) \sqsubseteq t_1^* \sqcap t_2^* \qquad (1)$$

where $SCC(t_1,t_2)$ is the smallest cube containing t_1 and t_2, and t_1^* and t_2^* are the overexpanded cubes of t_1 and t_2, respectively. Note that (1) is a necessary condition for t_1 and t_2 being covered by a prime implicant of F.

Lemma 3 Let C be a cover for function F, and G its associated covering graph. Assume that $t_1,t_2,\ldots,t_m$ are cubes in C and $v_1,v_2,\ldots,v_m$ are their corresponding vertices in G. Then the subgraph induced by $v_1,v_2,\ldots,v_m$ is a clique iff

$$SCC(t_1,t_2,\ldots,t_m) \sqsubseteq t_1^* \sqcap t_2^* \sqcap \ldots \sqcap t_m^* \qquad (2)$$

Proof: ($\Leftarrow$) If (2) holds, then

$$SCC(t_i,t_j) \sqsubseteq SCC(t_1,\ldots,t_m) \sqsubseteq t_1^* \sqcap \ldots \sqcap t_m^* \sqsubseteq t_i^* \sqcap t_j^*$$

for i,j = 1,2,...,m. By the definition of covering graph, there is an edge connecting v_i and v_j for any i and j. Thus $v_1,v_2,\ldots,v_m$ induces a clique in G.

($\Rightarrow$) If $v_1, v_2, \ldots, v_m$ induces a clique, then

$$SCC(t_i, t_j) \sqsubseteq t_i^* \sqcap t_j^* \text{ for any } i,j = 1,2,\ldots,m.$$

In other words, we have $t_i \sqsubseteq t_j^*$ for any i,j. Thus

$$t_i \sqsubseteq t_1^* \sqcap t_2^* \sqcap \cdots \sqcap t_m^* \text{ for any } i.$$

Since $SCC(t_1,\ldots,t_m)$ is the smallest cube containing $t_1,\ldots,t_m$, (2) must be true

Theorem 3 Let C be a cover for function F, and G its associated covering graph. Assume that $t_1, t_2, \ldots, t_m$ are cubes in C and $v_1, v_2, \ldots, v_m$ are their corresponding vertices in G. If $t_1, t_2, \ldots, t_m$ can be covered by a prime implicant of F, then $v_1, \ldots, v_m$ induces a clique in G.

Proof: If $t_1, \ldots, t_m$ can be covered by a prime implicant p, then

$$t_i \sqsubseteq p \sqsubseteq t_i^* \text{ for all } i = 1,2,\ldots,m$$

from the definition of overexpanded cube. Thus

$$SCC(t_1,\ldots,t_m) \sqsubseteq p \sqsubseteq t_1^* \sqcap \cdots \sqcap t_m^*.$$

This theorem then follows from Lemma 3.

Theorem 3 suggests that the expansion process be replaced by a covering process. Instead of expanding a cube to cover other cubes, the covering process is to generate a cube to cover a specified set of cubes which correspond to a clique in the covering graph. One advantage to this approach is that the covering procedure can be carried out within the cube $SCC(t_1,\ldots,t_m)$ while the expansion of a cube in MINI and ESPRESSO is done within an overexpanded cube. According to (2), $SCC(t_1,\ldots,t_m)$ is contained in any overexpanded cube. Furthermore, cubes in $SCC(t_1,\ldots,t_m)$ other than $t_1,\ldots,t_m$ need not be considered in the covering process. But the expansion process has to consider every cube contained in the overexpanded cube. Consequently the covering process considers fewer cubes than the expansion process does, so can be done more efficiently.

Let us illustrate the EXPAND procedure as follows:

Input: A cover C for function F.

Output: An expansion of C.

Algorithm EXPAND:

```
     Construct the covering graph G
     Loop until G contains no edge
          If G has a vertex $v_1$ of degree one
          then COVER($\{t_1,t_2\}$) where $t_1$ and $t_2$ are the
               cubes corresponding to $v_1$ and its neighbor respectively.
          else COVER(K) where K is a set of cubes in C which correspond
          to a maximal clique in G.
     Endloop
     EXPAND_INTO_PRIME
```

In this algorithm, the COVER procedure takes a set K of implicants as its input and computes an implicant which covers a maximal subset of K. Here K corresponds to a clique in the covering graph G. If COVER finds an implicant t which covers more than one implicant in K, then these implicants can be replaced by t in C, and at the same time their corresponding vertices in G are replaced by a new isolated vertex corresponding to t. On the other hand, if no implicant of F can cover two or more implicants in K, then we simply delete the edges of the clique which correspond to K. In this way an implicant in K which is not covered by t can still be covered later in the loop. Note that the COVER procedure does not generate a prime implicant in general. Thus we need the EXPAND_INTO_PRIME procedure which expands implicants one at a time into prime implicants. EXPAND_INTO_PRIME aims to expand implicants to as large as possible and its algorithm is similar to that in ESPRESSO.

The EXPAND algorithm is dynamic in nature. The cover C and the covering graph G are updated iteratively and reflect the state of the expansion. When G has a vextex v_1 of degree one with its neighbor v_2, we know that t_1 and t_2, the implicants corresponding to v_1 and v_2, should be replaced by $SCC(t_1,t_2)$ provided that $SCC(t_1,t_2)$ is an implicant of F. On the other hand, if G has no vertex of degree one, the algorithm identifies a maximal clique and tries to generate an implicant which covers as many implicants as possible. In this way, the order of expansion or covering is dynamic and can achieve the goal of expansion closely.

5. The IRREDUNDANT Procedure

Let C be a cover for function F. An implicant t in C is redundant iff C - $\{t\}$ is still a cover for F or equivalently $t \sqsubseteq \cup(C - \{t\}) \cup DC(F)$. An implicant t in C is called relatively essential or nonredundant if it is not redundant in C. A cover C is irredundant if every implicant in C is relatively essential. The IRREDUNDANT procedure aims to find a subcover of a cover which is not only irredundant but also minimal in size.

The simplest way to find an irredundant cover is to check the implicants one by one for redundancy; when a redundant implicant is detected, it is removed from the cover. This method is efficient but can hardly guarantee a minimal cover. On the other hand, ESPRESSO-MV maps the minimal cover problem into a set covering problem, and then uses heuristics to slove the set covering problem. This approach can generate a result closer to the optimum. However it seems that the set covering problem can grow to very large if there are a lot of redundant implicants. What is done in CUP is a tradeoff between the two extremes. We have designed a new algorithm for extracting an irredundant cover which is simple but still reasonably effective. The algorithm is based on the following lemma:

Lemma 4 Let C be a cover for function F, and let $t_1, t_2, \ldots, t_m$ be implicants of F in C. If each t_i is redundant in C and $t_i \sqcap t_j = \phi$ for all $i,j = 1,2,\ldots,m$, $i \neq j$, then C - $\{t_1, t_2, \ldots, t_m\}$ is a cover for F.

Proof: For any i, $i = 1,2,\ldots,m$, t_i is redundant in C. Thus

$$t_i \subseteq \cup(C - \{t_i\}) \cup DC(F).$$

Since $t_i \cap t_j = \phi$ for any $j \neq i$, we have

$$t_i \subseteq \cup(C - \{t_1,t_2,\ldots,t_m\}) \cup DC(F) \text{ for any } i.$$

By definition, $ON(F) \subseteq \cup C$

$$= \cup(C - \{t_1,t_2,\ldots,t_m\}) \cup t_1 \cup t_2 \cup \ldots \cup t_m$$

$$\subseteq \cup(C - \{t_1,t_2,\ldots,t_m\}) \cup DC(F)$$

Since $ON(F) \cap DC(F) = \phi$, we have

$$ON(F) \subseteq \cup(C - \{t_1,t_2,\ldots,t_m\}).$$

According to Lemma 4, we can construct a graph called the independence graph. Each vertex of the graph corresponds to a redundant implicant in C and two vertices are connected by an edge iff their corresponding implicants are disjoint. Then by identifying a maximal clique in the independence graph, we can identify a set of redundant implicants which can be removed from the cover C.

As in ESPRESSO, the IRREDUNDANT procedure in CUP distinguishes absolutely redundant implicants and partially redundant implicants. Let E be the set of relatively essential implicants in C. An implicant t is absolutely redundant if $t \in C - E$ and $t \subseteq \cup E \cup DC(F)$. An implicant in C is partially redundant if it is redundant but is not absolutely redundant. The IRREDUNDANT procedure identifies the sets of relatively essential implicants and partially redundant implicants E and R_p, and deletes an appropriate set of partially redundant implicants. Then the same procedure is repeated with respect to the new cover $E \cup R_p$.

Input: A cover C for function F

Output: A minimal and irredundant cover E for F.

Output: A minimal and irredundant cover E for F.

Algorithm IRREDUNDANT:

```
Compute the set E of relatively essential implicants in C
Compute the set $R_p$ of partially redundant implicants
While $R_p \neq \phi$ do
    Construct the indepence graph with respect to $R_p$
    Find a maximal clique in the independence graph, and remove
    its corresponding implicants from $R_p$
    For every implicant t in $R_p$
        if $t \nsubseteq \cup(E \cup R_p - \{t\}) \cup DC(F)$
        then $E = E \cup \{t\}$ and $R_p = R_p - \{t\}$
    for every implicant t in $R_p$
        if $t \subseteq \cup E \cup DC(F)$ then $R_p = R_p - \{t\}$
Endwhile.
```

6. The Unateness of a Function

In ESPRESSO-II the unateness of a Boolean function with binary inputs is exploited to speed up tautolgy checking and the REDUCE procedure. In this section we shall generalize the concept of unateness to a Boolean function with multiple-valued inputs. Let $f : P_1 \times P_2 \times \ldots \times P_n \rightarrow \{0,1\}$ be a Boolean function with multiple-valued inputs where $P_i = \{0, 1, \ldots, p_i-1\}$. f is monotone decreasing (increasing) in value j of variable i, $0 \leq j \leq p_i-1$, if there does not exist $X_1 \in P_1$, $X_2 \in P_2, \ldots, X_n \in P_n$ such that

$$f(X_1,X_2,\ldots,X_n) = 0 \ (1) \text{ and } f(X_1,\ldots,X_{i-1},j,X_{i+1},\ldots,X_n) = 1 \ (0).$$

A multiple-valued Boolean function which is monotone decreasing in one value of a variable X_i is said to be unate in X_i. A function which is unate in all of its variables is said to be a unate function.

Lemma 5 Given a Boolean function f with multiple-valued inputs. If f is monotone increasing in all values of variable X_i except value j, then f is monotone decreasing in value j of variable X_i.

Proof: If f is not monotone decreasing in value j, then there exists $\overline{X}_l \in P_l$, $l = 1,2,\ldots,n$ such that

$$f(\overline{X}_1, \overline{X}_2,\ldots,\overline{X}_n) = 0 \text{ and } f(\overline{X}_1,\ldots,\overline{X}_{i-1}, j, \overline{X}_{i+1},\ldots, \overline{X}_n) = 1.$$

This violates the assumption that f is monotone increasing in value $\overline{X}_i$ of variable X_i. Thus f must be monotone decreasing in value j of variable X_i.

In ESPRESSO-MV a function is defined to be unate in variable X_i if it is monotone increasing or decreasing in all the values of X_i. From Lemma 5 such a function must be monotone decreasing in one value of X_i. Thus the unateness defined in this paper is more general than that in ESPRESSO-MV.

Let C = $t_1, t_2, \ldots, t_k$ be a set of cubes where

$$t_h = S_{h1} \times S_{h2} \times \ldots \times S_{hn},\ h = 1,2,\ldots,k.$$

C is said to be unate in variable X_i if $\bigcup_{\substack{h=1,k \\ S_{hi} \neq P_i}} S_{hi} = P_i$.

Lemma 6. If C is a cover for function f and if C is unate in variable X_i, then f is unate in X_i.

Proof: Assume that C consists of cubes $t_1, t_2, \ldots, t_k$ as above. Let j be a value in $P_i - \bigcup_{\substack{h=1,k \\ S_{hi} \neq P_i}} S_{hi}$. We shall prove that f is monotone decreasing in value j of X_i. If this is not the case, then there exists $\bar{X}_l \in P_l$, l = 1,2,...,n, such that

$$f(\bar{X}_1,\ldots,\bar{X}_i,\ldots,\bar{X}_n) = 0 \text{ and } f(\bar{X}_1,\ldots,\bar{X}_{i-1},j,\bar{X}_{i+1},\ldots,\bar{X}_n) = 1.$$

Since $(\bar{X}_1,\ldots,\bar{X}_{i-1},j,\bar{X}_{i+1},\ldots,\bar{X}_n)$ is in the on-set of function f,

$$(\bar{X}_1,\ldots,\bar{X}_{i-1},j,\bar{X}_{i+1},\ldots,\bar{X}_n) \in t_g \text{ for some } g,\ g = 1,2,\ldots,k.$$

Considering the variable X_i in particular, we have $j \in S_{gi}$.

Since $j \notin \bigcup_{\substack{h=1,k \\ S_{hi} \neq P_i}} S_{hi}$ and $j \in S_{gi}$, we have $S_{gi} = P_i$. Consequently we obtain $(\bar{X}_1,\ldots,\bar{X}_{i-1},\bar{X}_i,\bar{X}_{i+1},\ldots,\bar{X}_n) \in t_g$ which contradicts

$f(\bar{X}_1,\ldots,\bar{X}_i,\ldots,\bar{X}_n) = 0$. Thus f is monotone decreasing in value j of X_i.

Lemma 7. If f is unate in variable X_i and if C is a prime cover for f, then C is unate in X_i.

Proof: Let $C = \{t_1, t_2, \ldots, t_k\}$ where $t_h = S_{h1} \times S_{h2} \times \ldots \times S_{hn}$, $h = 1,2,\ldots,h$. Assume that f is monotone decreasing in value j of X_i. We will show that $j \notin \bigcup_{\substack{h=1,k \\ S_{hi} \neq P_i}} S_{hi}$.

To the contrary, assume that $j \in S_{hi} \neq P_i$ for some $h = 1,2,\ldots,k$. Then for any $X_1 \in S_{h1}, \ldots, X_{i-1} \in S_{h,i-1}, X_{i+1} \in S_{h,i+1}, \ldots, X_n \in S_{hn}$, we have $(X_1, \ldots, X_{i-1}, j, X_{i+1}, \ldots X_n) \in t_h$. Thus

$$f(X_1, \ldots, X_{i-1}, j, X_{i+1}, \ldots, X_n) = 1 \text{ for any } X_l \in S_{hl},\ l \neq i.$$

Since f is monotone decreasing in value j,

$$f(X_1, \ldots, X_{i-1}, j, X_{i+1}, \ldots, X_n) = 1 \text{ for any } X_i \in P_i \text{ and } X_l \in S_{hl},\ l \neq i.$$

Thus $S_{h1} \times \ldots \times S_{h,i-1} \times P_i \times S_{h,i+1} \times \ldots \times S_{hn}$ is an implicant of f containing t_h. By assumption $S_{hi} \neq P_i$ contradicts that t_h is a prime implicant. Thus we have shown $j \notin \bigcup_{\substack{h=1,k \\ S_{hi} \neq P_i}} S_{hi}$

In other words, C is unate in X_i.

The unateness of a cover is useful in determining whether a function is a tautology. A Boolean function f is a tautology if $f(X_1, X_2, \ldots, X_n) = 1$ for all $X_i \in P_i$, $i = 1,2,\ldots,n$, and is denoted by $f \equiv 1$.

Lemma 8. Let C be a unate cover for function f. Then f is a tautology, iff C contains a cube $t = P_1 \times P_2 \times \ldots \times P_n$.

Proof: The if part is apparent.

($\Rightarrow$) Assume $f \equiv 1$.

Let $C = \{t_1, t_2, \ldots, t_k\}$ where $t_h = S_{h1} \times S_{h2} \times \ldots \times S_{hn}$, $h = 1,2,\ldots,k$.

Since C is unate, $\bigcup_{\substack{h=1,k \\ S_{hi} \neq P_i}} S_{hi} \neq P_i$ for any $i = 1,2,\ldots,n$.

Consider the minterm $\bar{t} = (X_1, X_2,\ldots, X_n)$ where $X_i \in P_i - \bigcup_{\substack{h=1,k \\ S_{hi} \neq P_i}} S_{hi}$. Since $f \equiv 1$, $\bar{t} \in t_h$ for some h, $h = 1,2,\ldots,k$. Then $X_i \in S_{hi}$ for any i. Since $X_i \notin \bigcup_{\substack{h=1,k \\ S_{hi} \neq P_i}} S_{hi}$ and $X_i \in S_{hi}$ for any i, $S_{hi} = P_i$ for any i, $i = 1,2,\ldots,n$. In other words, $t_h = P_1 X P_2 X \ldots X P_n$. This completes the proof.

Theorem 4. Let $C = \{t_1, t_2,\ldots, t_k\}$ be a cover which is unate in variables $X_1,\ldots, X_m$ where $1 \leq m \leq n$. Each cube t_h in C can be expressed as $t_h = u_h X v_h$ where u_h and v_h are cubes in variables $X_1,\ldots,X_m$ and $X_{m+1},\ldots, X_n$, respectively. Assume that

$U_h = P_1 X\ldots X P_m$ for $h = 1,\ldots,l$ and

$u_h = P_1 X\ldots X P_m$ for $h = l+1,\ldots,k$.

Then C is a tautology iff the subcover $\{t_1,\ldots,t_l\}$ is a tautology.

Proof: The if part is apparent.

($\Rightarrow$) Assume that C is a tautology. To the contrary, suppose that $\{t_1,\ldots,t_l\}$ is not a tautology. Then there exists a minterm (a,b) such that $(a,b) \notin t_h$ for any $h = 1,\ldots,l$ where a and b are minterms in variables $X_1,\ldots,X_m$ and $X_{m+1},\ldots,X_n$, respectively. Since $t_h = u_h X v_h$ and $u_h = P_1 X \ldots X P_m$ for $h = 1,\ldots,l$, $b \notin v_h$ for any $h = 1,\ldots,l$. On the other hand, since C is unate in variables $X_1,\ldots,X_m$, the cover $\{u_{l+1},\ldots,u_k\}$ is unate. From Lemma 8, $\{u_{l+1},\ldots, u_k\}$ is not a tautology. Thus there exists a minterm c in variables $X_1,\ldots,X_m$ such that $c \notin u_h$ for any $h = l+1,\ldots,k$. Consider the minterm (c,b) in variables $X_1,\ldots,X_n$. Then $(c,b) \notin t_h$ for $h = 1,\ldots,l$ since $b \notin v_h$, and $(c,b) \notin t_h$ for $h = l+1,\ldots,k$ since $c \notin u_h$. We have shown that (c,b) is not covered by any cube in C contradicting that C is a tautology. Thus $\{t_1,\ldots,t_l\}$ must be a tautology.

Reference

(1) P. Agrawal, V. D. Agrawal and N. N. Biswas, "Multiple Output Minimization", Proc. of the 22nd ACM-IEEE Design Automation Conference, PP.674-680, Jun 1985.

(2) Z. Arevalo and J. G. Bredeson, "A Method to Simplify a Boolean Function into a Near Minimal Sum-of-Products for Programmable Logic Arrays", IEEE Trans. on Comp., Vol. C-27, No.11, PP.1028-1039, Nov. 1978.

(3) R. K. Brayton et al.. "Fast Recursive Boolean Function Manipulation". Proc. of the 1982 ISCS P.85 May 1982.

(4) R. K. Brayton et al.."Logic Minimization Algorithms for VLSI Synthesis". Kluwer Academic Publishers, Boston, 1984.

(5) R. K. Brayton et al.."A Comparison of Logic Minimization Strategy Using ESPRESSO: An APL Program Package for Partitioned Logic Minimization". Proc. of ISCS, PP.42-48 May 1982.

(6) D. W. Brown, "A State-Machine Synthesizer-SMS" Proc. of the 18th ACM-IEEE Design Automation Conference PP. 301-305, June 1981.

(7) G. Caruso, "A Local Selection Algorithm for Switching Function Minimization" IEEE Trans. on Comp. Vol. C-33 No.1, Jan. 1984.

(8) M. R. Dagenais, V. K. Agarwal and N. C. Rumin " The McBOOLE Logic Minimizer", Proc. of the 22nd ACM-IEEE Design Automation Conference, PP.667-673. June 1985.

(9) L. Dietmeyer. Logic Design of Digital System,

(10) S. J. Hong, R. G. Cain and D. L. Ostapko, "Mini:A Heuristic Approach for Logic Minimization", IBM J. of Res. and Dev. Vol. 18, PP.443-458, Sep. 1974.

(11) J. F. Martinez-Earballido and V. M. Power, "PRONTO: QUICK PLA PRODUCT REDUCTION" Proc. of the 20th ACM-IEEE Design Automation Conference, PP.545-552, June 1983.

(12) E. Morreale, "Recursive Operators for Prime Implicant and Irredundant Normal Form Determination", IEEE Trans. on Computer, Vol. C-19, No.6, PP.504-509, June 1970.

(13) D. L. Ostapko and S. J. Hong, "Generating Test Examples for Heuristic Boolean Minimization", IBM J. of Res. & Dev. Vol. 18, PP.459-464, Sep. 1974.

(14) B. Reusch, "Generation of Prime Implicants from Subfunctions and a Unifying Approach to the Covering Problem", IEEE Trans. on Comp. Vol C-24, No.9, Sep. 1975.

(15) V. T. Rhyne, etl. "A New Technique for the Fast Minimization of Switching Functions", IEEE Trans. on Computer, Vol C-26, No.8, PP.757-764, Aug. 1977.

(16) R. L. Rudell and A. L. M. Sangiovanni-Vincentelli, "ESPRESSO-MV: Algorithms for Multiple-Valued Logic Minimization", Proc. of IEEE CICC PP.230-234. 1985.

(17) T. Sasao, "Multiple-Valued Decomposition of Generalized Boolean Functions and The Complexity of Programmable Logic Arrays", IEEE Trans. on Comp. Vol C-30, No.9, PP.635-643, Sep. 1981.

(18) T. Sasao, " A Fast Complementation Algorithm for Sum-of-Products Expressions of Multiple-Valued Input Binary Functions", Proc. of 13th ISMVL, PP.103-110 May 1983.

(19) T. Sasao, "Tautology Checking Algorithms for Multiple-Valued Input Binary Functions and Their Application", Proc. of 14th ISMVL, PP.242-250, May 1984.

(20) T. Sasao, "Input Variable Assignment and Output Phase Optimization of PLA's", IEEE Trans. on Comp., Vol. C-33, No.10, Oct. 1984, PP.879-894.

(21) B. Teel and D. Wilde, "A Logic Minimizer for VLSI PLA Design", Proc. of the 19th ACM-IEEE Design Automation Conference, PP.156-162, May 1982.

Generating Essential Primes for a Boolean Function with Multiple-Valued Inputs

Y. S. Kuo and W. K. Chou

Institute of Information Science
Academia Sinica
Taiwan, Republic of China

Address for correspondence

Dr. Y. S. Kuo
Institute of Information Science
Academia Sinica
Taipei 11529, Taiwan
Republic of China

Tel: (886) 02-782-4814

Abstract

Detecting essential primes is important in multiple-valued logic minimization. In this paper, We present a fast algorithm that can generate all essential primes without generating a prime cover of the Boolean function. A new consensus operation called asymmetric consensus (acons) is defined. In terms of acons, we prove a necessary and sufficient condition for detecting essential primes for a Boolean function with multiple-valued inputs. The detection of essential primes can be performed by using a tautology checking algorithm. We exploit the unateness of a Boolean function to speed up tautology checking. The notion of unateness considered is more general than that has appeared in the literature.

1. Introduction

Boolean functions with multiple-valued inputs have gained more and more attention due to their application to the design of programmable logic arrays (PLAs). In principle, a PLA can implement any multiple-output switching function. Such a switching function can be treated as a Boolean function with multiple-valued inputs; the outputs of the switching function are treated as one multiple-valued input of the Boolean function [7]. Then minimizing the number of rows of a PLA is equivalent to minimizing the number of product terms in the sum-of-products expression for the Boolean function. Boolean functions with multiple-valued inputs can be further exploited in designing PLAs with decoders [2] [5]. A minimized PLA with t-bit decoders corresponds to a minimum sum-of-products expression for a Boolean function with 2^t-valued inputs.

Given a Boolean function f with multiple-valued inputs. An essential prime implicant (or simply essential prime) is a prime implicant of f which contains a minterm in the on-set of f that is not contained in any other prime implicant of f. Generating essential primes is important in 2-level logic minimization [1] [3] since every essential prime must appear in a minimum sum-of-products expression for a Boolean function. It has been reported that Boolean functions for the control circuits of microprocessors often contain a large number of essential primes [6]. Therefore, we can obtain near-optimal expressions quickly if we can detect all essential primes at early stages of the logic manipulation process.

Conventionally, essential primes are generated by first generating a prime cover for the Boolean function (or even generating all prime implicants). Take the well-known logic minimizer ESPRESSO-II [1] as an example. This system has a procedure EXPAND that expands implicants into prime implicants and thus can generate a prime cover for the Boolean function. The detection of essential primes in ESPRESSO-II is performed after the EXPAND procedure has generated a prime cover. In this paper, however, we shall argue that it is advantageous to generate essential primes before a prime cover is found. A method is presented which can generate all essential primes without generating a prime cover. Compared with the conventional EXPAND-then-essential-primes approach, this essential-primes-then-EXPAND approach causes very little computation overhead and in many cases can even speed up the EXPAND procedure.

Given a cover for a Boolean function with multiple-valued inputs. The essential primes can be generated by first generating the overexpanded cubes [3] of those implicants in the cover. An overexpanded cube is a potential essential prime if it is an implicant of the Boolean function. Such an overexpanded cube will be called a partially essential prime. Then a test is performed to check whether the overexpanded cube is actually an essential prime. A necessary and sufficient condition for detecting essential primes for a Boolean function with binary inputs has been known for some time [1]. In the case of multiple-valued inputs, Sasao gave a sufficient condition for detecting essential primes [6] [7]. One step further, we shall prove a necessary and sufficient condition for essential prime detection in the multiple-valued case. This condition has been stated in terms of a new consensus operation called the asymmetric consensus, and can be tested by using a tautology checking algorithm [1] [6].

Tautology checking is a basic procedure used extensively in logic minimization, and detecting essential primes is one of its applications. In ESPRESSO-II (binary inputs) [1] and ESPRESSO-MV (multiple-valued inputs) [4] properties of unate functions have been exploited to speed up the tautology checking algorithm. In this paper, we shall consider a more general notion of unateness and show that the same results as those for ESPRESSO-MV [4] still hold. Thus the tautology checking algorithm can be further improved in speed.

In Section 2 we introduce the notation for multilple-valued logic minimization. Different consensus operations are defined and compared. In Section 3 an algorithm for generating essential primes is described which does not generate a prime cover. The relationship between this algorithm and the EXPAND procedure is also investigated. Then we prove a necessary and sufficient condition for detecting essential primes in Section 4. In Section 5 the unateness of a Boolean function is exploited for tautology checking. Finally we try to draw some conclusions in the last section.

2. Notation

Let us define some concepts in multiple-valued logic. A Boolean function with multiple-valued inputs is a mapping

$$f: P_1 \times P_2 \times \dots \times P_n \longrightarrow \{0,1\}$$

where $P_i = \{0,1,\dots,p_i-1\}$. A generalized Boolean function with multiple-valued inputs is a mapping

$$F: P_1 \times P_2 \times \dots \times P_n \longrightarrow \{0,1,d\}.$$

Each element of $P_1 \times P_2 \times \dots \times P_n$ is called a minterm. The on-set, off-set and dc-set of a (generalized) Boolean function F are the sets of minterms that have function values 1,0 and d, and will be denoted by ON(F), OFF(F) and DC(F), respectively. Note that a Boolean function can be characterized by its on-set and vice versa.

A cube is a Cartesian product $S_1 \times S_2 \times \dots \times S_n$ where $S_i \subseteq P_i$ for all $i = 1,2,\dots,n$. The Boolean function associated with a cube is a product term. A cube t is an implicant of a (generalized) Boolean function F if $t \cap \mathrm{OFF}(F) = \phi$. t is called a prime implicant if t is an implicant of F and there is no implicant t' of F with $t \subseteq t'$ and $t' \neq t$.

Let C be a set of implicants of function F. Then C is called a cover for F if every minterm in the on-set of F is covered by an implicant in C. If every implicant of cover C is a prime implicant, then C is called a prime cover. Note that a cover for function F corresponds to a sum-of-products expression for F. Thus the (two-level) logic minimization problem is to find a prime cover for a generalized Boolean function of minimum size.

Two defferent consensus operations have been used for detecting essential primes [1] [7]. But we shall introduce a third one called the asymmetric consensus. In the following these operations will be defined and then compared.

Let t_1 and t_2 be cubes where

$t_1 = S_1 \times S_2 \times \ldots \times S_n$ and $t_2 = R_1 \times R_2 \times \ldots \times R_n$.

The consensus of t_1 and t_2 is

$$\text{cons}\ (t_1,t_2) = \begin{cases} t_1 \cap t_2 & \text{if } t_1 \cap t_2 \neq \phi\ , \\ \bigcup\limits_{i=1,n} (S_1 \cap R_1) \times \ldots \times (S_i \cup R_i) \times \ldots \times (S_n \cap R_n) & \text{otherwise.} \end{cases}$$

The union consensus of t_1 and t_2 is

$$\text{ucons}\ (t_1,t_2) = \bigcup_{i=1,n} (S_1 \cap R_1) \times \ldots \times (S_i \cup R_i) \times \ldots \times (S_n \cap R_n).$$

The asymmetric consensus of t_1 and t_2 is

$$\text{acons}\ (t_1,t_2) = \bigcup_{\substack{i=1,n \\ S_i \cup R_i \neq R_i}} (S_1 \cap R_1) \times \ldots \times (S_i \cup R_i) \times \ldots \times (S_n \cap R_n).$$

Lemma 1 If $t_1 \cap t_2 = \phi$, then

cons (t_1,t_2) = ucons (t_1,t_2) = acons (t_1,t_2).

Moreover, if t_1 and t_2 are disjoint in more than one variable, then

cons (t_1,t_2) = ucons (t_1,t_2) = acons $(t_1,t_2) = \phi$.

Proof: Omitted.

Lemma 2 acons $(t_1,t_2) \subseteq$ ucons (t_1,t_2).

If $t_1 \nsubseteq t_2$, then cons $(t_1,t_2) \subseteq$ acons (t_1,t_2).

Proof: Omitted.

Unlike cons and ucons, acons is not symmetric, i.e. acons $(t_1,t_2) \neq$

acons (t_2,t_1) in general. Let us illustrate the differences among these operations by an example.

Example 1 The 3-output switching function represented by the Karnaugh maps in Fig.1 can be treated as a Boolean function defined on $\{0,1\} \times \{0,1\} \times \{0,1,2\}$. This function consists of cubes t_1 and t_2 where

$$t_1 = \{0\} \times \{1\} \times \{0,1,2\} \text{ and } t_2 = \{0,1\} \times \{1\} \times \{1,2\}.$$

Then cons $(t_1,t_2) = \{0\} \times \{1\} \times \{1,2\}$

$$\begin{aligned} \text{ucons } (t_1,t_2) &= \{0,1\} \times \{1\} \times \{1,2\} \cup \{0\} \times \{1\} \times \{1,2\} \\ &\quad \cup \{0\} \times \{1\} \times \{0,1,2\} \\ &= \{0,1\} \times \{1\} \times \{1,2\} \cup \{0\} \times \{1\} \times \{0,1,2\} \end{aligned}$$

$$\text{acons } (t_1,t_2) = \{0\} \times \{1\} \times \{0,1,2\}$$

$$\text{acons } (t_2,t_1) = \{0,1\} \times \{1\} \times \{1,2\}.$$

Lemma 3 Let # denote the sharp operation, i.e. $t_1 \# t_2 = \bigcup_{\substack{i=1,n \\ S_i - R_i \neq \phi}} h_i$

where $h_i = S_1 \times \dots \times S_{i-1} \times (S_i - R_i) \times S_{i+1} \times \dots \times S_n$. Then

$$\text{acons } (t_1,t_2) = \bigcup_{\substack{i=1,n \\ S_i - R_i \neq \phi}} \text{acons } (h_i,t_2).$$

Proof: If $t_1 \cap t_2 = \phi$, the result is apparently true. Thus we assume that $t_1 \cap t_2 \neq \phi$. Since h_i and t_2 intersect in all variables except variable x_i, we have

$$\begin{aligned} \text{acons } (h_i,t_2) &= (S_1 \cap R_1) \times \dots \times (S_{i-1} \cap R_{i-1}) \times ((S_i - R_i) \cup R_i) \times \\ &\quad (S_{i+1} \cap R_{i+1}) \times \dots \times (S_n \cap R_n) \\ &= (S_1 \cap R_1) \times \dots \times (S_{i-1} \cap R_{i-1}) \times (S_i \cup R_i) \times \\ &\quad (S_{i+1} \cap R_{i+1}) \times \dots \times (S_n \cap R_n). \end{aligned}$$

This Lemma then follows from the definition of acons.

3. Generating Essential Primes

Given a generalized Boolean function F. A prime implicant p of F is an essential prime if p contains a minterm x in the on-set of F which is not contained in any other prime implicant of F. The minterm x is referred to as a distinguished minterm.

It is well-known that any prime cover for a Boolean function contains all essential primes. Thus in most logic minimization systems, the detection of essential primes is performed after a prime cover has been generated. But this can lead to pitfalls as demonstrated by the following example.

Example 2 The Karnaugh map of a 4-input Boolean function is shown in Fig.2. Assume that the initial cover for the function consists of minterms. If the minterm $\bar{A}B\bar{C}D$ is expanded first, it is likely that it will be expanded into $\bar{A}\bar{C}D$. Then when $\bar{A}BCD$ or $ABCD$ is expanded, the product term BCD can be generated. Eventually we have the prime cover $\{\bar{A}\bar{C}D, BCD, \bar{A}\bar{B}\bar{C}, ABC\}$. Even though we can find out that $\bar{A}\bar{B}\bar{C}$ and ABC are essential primes, we still have a prime cover of size 4 while the minimum cover size is 3. However, if we can detect the essential primes $\bar{A}\bar{B}\bar{C}$ and ABC before the expansion, then those cubes covered by the essential primes can be excluded from the current cover. Consequently we can obtain the minimum prime cover $\{\bar{A}\bar{B}\bar{C}, ABC, \bar{A}BD\}$.

Let us describe a procedure that can generate all essential primes without generating a prime cover. This procedure can serve as a preprocess for the EXPAND procedure mentioned in Section 1, so will be referred to as PRE_EXPAND.

Let C be a cover for a Boolean function F with multiple-valued inputs. The PRE_EXPAND procedure computes the overexpanded cubes t^* for implicants t in C [3]. Note that the overexpanded cube t^* of t is the smallest cube containing all prime implicants of F which contain t. If the off-set of F is known, then t^* can be computed easily [3] [8]. The PRE_EXPAND procedure also checks whether $t^* \cap \mathrm{OFF}(F) = \phi$ or not.

Lemma 4 If $t^* \cap \mathrm{OFF}(F) = \phi$, then t^* is a prime implicant of F.
Proof: Since $t^* \cap \mathrm{OFF} = \phi$, t^* is an implicant of F. From the definition of overexpanded cube, t^* contains at least a prime implicant. Thus t^* must be a prime implicant itself.

An overexpanded cube t^* satisfying $t^* \cap \mathrm{OFF} = \phi$ will be called a partially essential prime (with respect to cover C). Let $A = \{ t^* \mid t \in C, t^* \cap \mathrm{OFF} = \phi \}$ be the set of partially essential primes. Then A contains all essential primes.

Theorem 1 Let C be a cover for function F. If p is any essential prime of F, then $p = t^*$ for some $t \in C$.
Proof: Let x be a distinguished minterm contained in p. Since x is in the on-set of F, $x \in t$ for some $t \in C$. Then $t \subseteq p$, otherwise t is contained in a prime implicant other than p contradicting the fact that x is a distinguished minterm. Since p is the only prime implicant containing x, p is the only prime implicant containing t. Thus from the definition of overexpanded cube, $t^* = p$.

In Theorem 1, $t^* \cap$ OFF $= \phi$ apparently. The following result provides a quick check for essential primes.

Lemma 5 If $t^* \cap$ OFF $= \phi$ and t is a minterm in the on-set of F, then t^* is an essential prime of F.

Proof: From Lemma 4, t^* is a prime implicant. Actually t^* is the only prime implicant containing t. Thus t is a distinguished minterm contained in t^*.

Let us demonstrate the algorithm for PRE_EXPAND.

Input: A cover C for a generalized Boolean function F.

Output: A cover C for F which contains all partially essential primes.

Algorithm PRE_EXPAND:

```
for each $t \in C$ do
    compute the overexpanded cube $t^*$
    if $t^* \cap$ OFF $= \phi$ then
        replace t in C by $t^*$
        delete all cubes in C that are contained in $t^*$
    endif
endfor
```

Even though PRE_EXPAND does not compute all overexpanded cubes in general, it does generate all partially essential primes. An algorithm that can detect essential primes among these partially essential primes will be described in the next section. For the rest of this section we investigate the relationship between the PRE_EXPAND and the EXPAND procedures.

Given a cover C for a generalized Boolean function F. A prime cover E for F is called an expansion of C if for any implicant t in C, there is a prime implicant t' in E containing t. According to this definition, the EXPAND procedure can generate an expansion of a cover and its objective is to generate an expansion of minimum size.

Lemma 6 Any expansion E of C contains all partially essential primes, if any.

Proof: If t^* is an arbitrary partially essential prime with $t \in C$, then t^* is the only prime implicant containing t. Since E must contain a prime implicant containing t, we have $t^* \in E$. This completes the proof.

Due to Lemma 6 the PRE_EXPAND procedure can be considered as a preprocess for the EXPAND procedure. It preprocesses certain cubes so that part of a minimum expansion can be generated.

Example 3 The Boolean function represented by the Karnaugh map in Fig.3 has no essential primes. This function has a cover $C = \{ t_1, t_2, \ldots, t_6 \}$ where $t_1 = \bar{A}D$, $t_2 = B\bar{C}\bar{D}$, $t_3 = AB\bar{C}D$, $t_4 = A\bar{B}\bar{C}$, $t_5 = A\bar{B}C$ and $t_6 = \bar{A}C\bar{D}$. Then $t_1^* = \bar{A}D$, $t_2^* = B\bar{C}$, $t_3^* = \bar{C}$, $t_4^* = A$, $t_5^* = \bar{B}$ and $t_6^* = \bar{A}C$. Thus t_1^*, t_2^* and t_6^* are partially essential primes, and the PRE_EXPAND procedure generates the cover $\{ t_1^*, t_2^*, t_6^*, t_4, t_5 \}$. After this preprocessing, the EXPAND procedure can generate the prime cover $\{ t_1^*, t_2^*, t_6^*, A\bar{B} \}$ easily.

In contrast, if PRE_EXPAND were not applied first, the EXPAND procedure could expand t_3 or t_4 before t_2. Then it is quite possible that the product term $A\bar{C}$ would be generated. Eventually EXPAND would generate the prime cover $\{A\bar{C}, t_1^*, t_2^*, t_6^*, A\bar{B}\}$ or $\{A\bar{C}, t_1^*, t_2^*, t_6^*, \bar{B}C\}$, either of which is of size 5.

4. Detecting Essential Primes

Given a cover C for a generalized Boolean function F. Assume that p is a prime implicant of F in C. We want to determine whether p is an essential prime or not.

Let us make some definitions.

Let D be a DC-cover for function F, i.e. D is a set of cubes satisfying $\bigcup_{t \in D} t = DC(F)$. Define $G = (C - \{p\}) \cup D$ as the set of cubes in C or D other than p, and define

$$H = \bigcup_{h \in G} \text{acons}(h,p)$$
$$H_1 = \bigcup_{h \in G} \text{cons}(h,p)$$
$$H_2 = \bigcup_{h \in G} \text{ucons}(h,p).$$

First, let us consider two relevant known results [1] [7].

Lemma 7 Let F be a generalized Boolean function with binary inputs. Assume that p contains no cube in $C - \{p\}$. Then p is an essential prime iff $p \cap ON(F) \not\subseteq H_1$.

Lemma 8 Let F be a Boolean function with multiple-valued inputs. If $p \not\subseteq H_2$, then p is an essential prime.

It should be noted that Lemma 7 holds only for binary-input, single-output switching functions even though it was claimed to hold for binary-input, multiple-output switching functions [1]. On the other hand, Lemma 8 gives a necessary condition for detecting essential primes, but not a sufficient condition.

Example 4 consider the binary-input, 3-output switching function represented by the Karnaugh maps in Fig.4. This function can be treated as a Boolean function defined on $\{0,1\} \times \{0,1\} \times \{0,1\} \times \{0,1,2\}$ and has a prime cover consisting of cubes

$$t_1 = \{0\} \times \{0\} \times \{0,1\} \times \{1,2\}$$
$$t_2 = \{0,1\} \times \{0\} \times \{0\} \times \{0,1\}$$
and $t_3 = \{0\} \times \{0,1\} \times \{1\} \times \{1,2\}$.

To check whether t_1 is essential, we compute H_1 with respect to t_1.

$$H_1 = \text{cons } (t_2,t_1) \cup \text{cons } (t_3,t_1)$$
$$= \{0\} \times \{0\} \times \{0\} \times \{1\} \cup \{0\} \times \{0\} \times \{1\} \times \{1,2\}$$

Note that $t_1 \nsubseteq H_1$ but t_1 is not an essential prime. Thus Lemma 7 is not true for binary-input, multiple-output switching functions.

To check whether t_3 is essential, we compute H_2 with respect to t_3.

$$H_2 = \text{ucons } (t_1,t_3) \cup \text{ucons } (t_2,t_3)$$
$$= \{0\} \times \{0,1\} \times \{1\} \times \{1,2\} \cup$$
$$\{0\} \times \{0\} \times \{0,1\} \times \{1,2\}$$

Here $t_3 \subseteq H_2$, but t_3 is an essential prime. Thus the converse of Lemma 8 is not true.

As shown in Example 4, cons (H_1) and ucons (H_2) can not provide a necessary and sufficient condition for detecting essential primes in multiple-valued logic. Thus we consider the new acons operation. We shall prove that p is essential iff $p \cap ON(F) \nsubseteq H$.

Lemma 9 Given a generalized Boolean function F with multiple-valued inputs. Let C be a cover for F and p a prime implicant of F in C. Assume that D, G and H are defined as in the beginning of this section, and assume that $p \cap h = \phi$ for all $h \in G$.

Then p is an essential prime iff $p \cap ON(F) \nsubseteq H$.

Proof:

($\Rightarrow$) Assume that p is essential.

Let x be a distinguished minterm in p. Suppose $x \in H$, i.e.

$$x \in \text{acons}\ (h,p) \text{ for some } h \in G.$$

Let $p = R_1 \times R_2 \times \ldots \times R_n$ and $h = S_1 \times S_2 \times \ldots \times S_n$. Then from assumption, $p \cap h = \phi$ and

$$\text{acons}\ (h,p) = (S_1 \cap R_1) \times \ldots \times (S_i \cup R_i) \times \ldots \times (S_n \cap R_n)$$

where $S_i \cap R_i = \phi$ and $S_j \cap R_j \neq \phi$ for $j \neq i$.

Let $x = (x_1, x_2, \ldots, x_n)$. Then

$$x_j \in S_j \cap R_j \text{ for all } j \neq i,\ j = 1,2,\ldots,n.$$

Consider a minterm $x' = (x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots x_n)$ where $y \in S_i$.

Then $x' \in \text{acons}(h,p)$ but $x' \notin p$. We have shown

$$x \in \text{acons}(h,p) \nsubseteq p.$$

This contradicts the assumption that x is a distinguished minterm. Thus $x \notin H$. Since $x \in p \cap ON$, $p \cap ON \nsubseteq H$.

($\Leftarrow$) Assume that p is not an essential prime.

Then for any minterm $x \in p \cap ON$, there exists an implicant t of F such that $x \in t$ and $t \nsubseteq p$. Let

$$p = R_1 \times R_2 \times \ldots \times R_n \text{ and } t = S_1 \times S_2 \times \ldots \times S_n.$$

Then $S_i \nsubseteq R_i$ for some i. Let $x = (x_1, x_2, \ldots, x_n)$. Since $x \in p \cap t$, $x_j \in R_j \cap S_j$ for all $j = 1,2,\ldots,n$.

Consider a minterm $x' = (x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n)$ where $y \in S_i - R_i$.

Then $x' \in t$ and $x' \notin p$. Since t is an implicant of F and $x' \in t$, there exists a cube $h \in G$ containing x'.

Let $h = T_1 \times T_2 \times \ldots \times T_n$. By assumption, $p \cap h = \phi$.

But $R_j \cap T_j \neq \emptyset$ for all $j \neq i$ because $x \in p$ and $x' \in h$. Thus

$$acons(h,p) = (T_1 \cap R_1) \times \ldots \times (T_i \cup R_i) \times \ldots \times (T_n \cap R_n).$$

Since $x_j \in T_j \cap R_j$ for all $j \neq i$ and $x_i \in R_i$, $x \in acons(h,p)$.

Since x is an arbitrary minterm in $p \cap ON$, we have $p \cap ON \subseteq H$. This completes the proof.

Theorem 2 Let C be a cover for a generalized Boolean function F with multiple-valued inputs, and let p be a prime implicant of F in C. Let D be a DC-cover for F and define

$$G = (C - \{p\}) \cup D,$$

$$H = \bigcup_{h \in G} acons\,(h,p).$$

Then p is an essential prime iff $p \cap ON(F) \nsubseteq H$.

Proof: We can construct a new cover C' and a new DC-cover D' such that they cover the same sets of minterms as C and D, respectively, but

$$p \cap h' = \emptyset \quad \text{for all } h' \in G'$$

where $G' = (C' - \{p\}) \cup D'$. This construction is done by taking a sharp operation on every cube h in G with $h \cap p \neq \emptyset$. Formally, C' is defined as follows: $p \in C'$; if $h \in C - \{p\}$ and $h \cap p = \emptyset$, then $h \in C'$; if $h \in C - \{p\}$ but $h \cap p \neq \emptyset$, then every $h_i \in C'$ where $h \# p = \bigcup h_i$. D' can be constructed from D similarly.

It should be noted that whether p is an essential prime is determined by the on-, off- and dc-sets (of minterms), not by the covers for these sets. Therefore we can apply Lemma 9 to C' and D'. Then p is an essential prime iff $p \cap ON \nsubseteq H'$ where $H' = \bigcup_{h' \in G'} acons\,(h',p)$.

From Lemma 3 we know that H and H' cover the same set of minterms. Thus p is essential iff $p \cap ON \nsubseteq H$.

In Theorem 2 we have to check whether $p \cap ON \subseteq H$ or not. It has been shown that this can be done with a tautology checking algorithm [1] [6].

Example 5 Consider the Boolean function and its prime cover $\{t_1, t_2, t_3\}$ of Example 4. To check whether t_1 is essential, we compute H with respect to t_1.

$$H = \text{acons}\ (t_2, t_1) \cup \text{acons}\ (t_3, t_1)$$
$$= \{0,1\} \times \{0\} \times \{0\} \times \{1\} \cup \{0\} \times \{0\} \times \{0\} \times \{0,1,2\} \cup$$
$$\{0\} \times \{0,1\} \times \{1\} \times \{1,2\} .$$

Since $t_1 \subseteq H$, from Theorem 2 t_1 is not an essential prime.

To check whether t_3 is essential, we compute H with respect to t_3.

$$H = \text{acons}\ (t_1, t_3) \cup \text{acons}\ (t_2, t_3)$$
$$= \{0\} \times \{0\} \times \{0,1\} \times \{1,2\} .$$

Since $t_3 \nsubseteq H$, from Theorem 2 t_3 is an essential prime.

5. The Unateness of a Boolean Function

In ESPRESSO-II [1] the unateness of a Boolean function with binary inputs is exploited to speed up the tautolgy checking algorithm. In this section we shall generalize the concept of unateness to a Boolean function with multiple-valued inputs.

Let $f : P_1 \times P_2 \times \ldots \times P_n \longrightarrow \{0,1\}$ be a Boolean function with multiple-valued inputs where $P_i = \{0, 1, \ldots, p_i-1\}$. Then f is monotone decreasing (increasing) in value j of variable x_i, $0 \le j \le p_i-1$, if there does not exist $\bar{x}_1 \in P_1, \bar{x}_2 \in P_2, \ldots, \bar{x}_n \in P_n$ such that

$f(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n) = 0$ (1) and $f(\bar{x}_1, \ldots, \bar{x}_{i-1}, j, \bar{x}_{i+1}, \ldots, \bar{x}_n) = 1$ (0).

A multiple-valued Boolean function which is monotone decreasing in at least a value of a variable x_i is said to be unate in x_i. A function which is unate in all of its variables is said to be a unate function.

Lemma 10 Given a Boolean function f with multiple-valued inputs. If f is monotone increasing in all values of variable x_i except value j, then f is monotone decreasing in value j of variable x_i.

Proof: If f is not monotone decreasing in value j, then there exists $\bar{x}_e \in P_e$, $e = 1,2,\ldots,n$ such that

$f(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n) = 0$ and $f(\bar{x}_1, \ldots, \bar{x}_{i-1}, j, \bar{x}_{i+1}, \ldots, \bar{x}_n) = 1$.

This violates the assumption that f is monotone increasing in value $\bar{x}_i$ of variable x_i. Thus f must be monotone decreasing in value j of variable x_i.

In ESPRESSO-MV [4] a function is defined to be unate in variable x_i if it is monotone increasing or decreasing in all the values of x_i.

From Lemma 10 such a function must be monotone decreasing in one value of x_i. Thus the unateness defined in this paper is more general than that in ESPRESSO-MV.

Let $C = \{t_1, t_2, \ldots, t_k\}$ be a set of cubes, i.e. a cover, where

$$t_h = S_{h1} \times S_{h2} \times \ldots \times S_{hn},\ h = 1,2,\ldots,k.$$

Then C is said to be unate in variable x_i if $\bigcup_{\substack{h=1,k \\ S_{hi} \neq P_i}} S_{hi} \neq P_i$.

C is called a unate cover if C is unate in all variables x_i, $i = 1,2,\ldots,n$.

Lemma 11. If C is a cover for function f and if C is unate in variable x_i, then f is unate in x_i.

Proof: Assume that C consists of cubes $t_1, t_2, \ldots, t_k$ as above. Let $Q_i = \bigcup_{\substack{h=1,k \\ S_{hi} \neq P_i}} S_{hi}$, and let j be a value in $P_i - Q_i$. We shall prove that f is monotone decreasing in value j of x_i. If this is not the case, then there exists $\bar{x}_e \in P_e$, $e = 1,2,\ldots,n$, such that

$$f(\bar{x}_1,\ldots,\bar{x}_i,\ldots,\bar{x}_n) = 0 \text{ and } f(\bar{x}_1,\ldots,\bar{x}_{i-1},j,\bar{x}_{i+1},\ldots,\bar{x}_n) = 1.$$

Since $(\bar{x}_1,\ldots,\bar{x}_{i-1},j,\bar{x}_{i+1},\ldots,\bar{x}_n)$ is in the on-set of function f,

$$(\bar{x}_1,\ldots,\bar{x}_{i-1},j,\bar{x}_{i+1},\ldots,\bar{x}_n) \in t_g \text{ for some } g,\ g \in \{1,2,\ldots,k\}.$$

Considering the variable x_i in particular, we have $j \in S_{gi}$.

Since $j \notin Q_i$ and $j \in S_{gi}$, we have $S_{gi} = P_i$. Consequently we obtain

$$(\bar{x}_1,\ldots,\bar{x}_{i-1},\bar{x}_i,\bar{x}_{i+1},\ldots,\bar{x}_n) \in t_g$$

which contradicts $f(\bar{x}_1,\ldots,\bar{x}_i,\ldots,\bar{x}_n) = 0$. Thus f is monotone decreasing in value j of x_i.

Lemma 12. If f is unate in variable x_i and if C is a prime cover for f, then C is unate in x_i.

Proof: Let $C = \{t_1, t_2, \ldots, t_k\}$ where $t_h = S_{h1} \times S_{h2} \times \ldots \times S_{hn}$, $h = 1,2,\ldots,k$. Assume that f is monotone decreasing in value j of x_i. We will show that $j \notin Q_i$ where $Q_i = \bigcup_{\substack{h=1,k \\ S_{hi} \neq P_i}} S_{hi}$.

To the contrary, assume that $j \in S_{hi} \neq P_i$ for some $h \in \{1,2,\ldots,k\}$. Then for any $\bar{x}_1 \in S_{h1}, \ldots, \bar{x}_{i-1} \in S_{h,i-1}$, $\bar{x}_{i+1} \in S_{h,i+1}, \ldots, \bar{x}_n \in S_{hn}$, we have $(\bar{x}_1, \ldots, \bar{x}_{i-1}, j, \bar{x}_{i+1}, \ldots \bar{x}_n) \in t_h$. Thus

$$f(\bar{x}_1, \ldots, \bar{x}_{i-1}, j, \bar{x}_{i+1}, \ldots, \bar{x}_n) = 1 \text{ for any } \bar{x}_e \in S_{he},\ e \neq i.$$

Since f is monotone decreasing in value j,

$$f(\bar{x}_1, \ldots, \bar{x}_{i-1}, \bar{x}_i, \bar{x}_{i+1}, \ldots, \bar{x}_n) = 1 \text{ for any } \bar{x}_i \in P_i \text{ and } \bar{x}_e \in S_{he},\ e \neq i.$$

Thus $S_{h1} \times \ldots \times S_{h,i-1} \times P_i \times S_{h,i+1} \times \ldots \times S_{hn}$ is an implicant of f containing t_h. By assumption $S_{hi} \neq P_i$, this contradicts that t_h is a prime implicant. Thus we have shown $j \notin Q_i$. In other words, C is unate in x_i.

The unateness of a cover is useful in determining whether a function is a tautology. A Boolean function f is a tautology if $f(x) = 1$ for any minterm $x \in P_1 \times P_2 \times \ldots \times P_n$. A cover C is a tautology if it covers every minterm of $P_1 \times P_2 \times \ldots \times P_n$.

Lemma 13. Let C be a unate cover. Then C is a tautology iff C contains the cube $P_1 \times P_2 \times \ldots \times P_n$.

Proof: The if part is apparent.

($\Rightarrow$) Assume that C is a tautology.

Let $C = \{t_1, t_2, \ldots, t_k\}$ where $t_h = S_{h1} \times S_{h2} \times \ldots \times S_{hn}$, $h = 1,2,\ldots,k$.

Let $Q_i = \bigcup_{\substack{h=1,k \\ S_{hi} \neq P_i}} S_{hi}$. Since C is unate, $Q_i \neq P_i$ for any $i = 1,2,\ldots,n$.

Consider the minterm $x = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$ where $\bar{x}_i \in P_i - Q_i$. Since C is a tautology, $x \in t_g$ for some g, $g \in \{1,2,\ldots,k\}$.

Then $\bar{x}_i \in S_{gi}$ for any $i = 1,2,\ldots,n$. Since $\bar{x}_i \notin \bigcup_{\substack{i=1,k \\ S_{hi} \neq P_i}} S_{hi}$ and $\bar{x}_i \in S_{gi}$ for any i, we have $S_{gi} = P_i$ for any i, $i = 1,2,\ldots,n$. In other words, $t_g = P_1 \times P_2 \times \ldots \times P_n$. This completes the proof.

Theorem 3. Let $C = \{t_1, t_2, \ldots, t_k\}$ be a cover which is unate in variables $x_1, \ldots, x_m$ where $1 \leq m \leq n$. Each cube t_h in C can be expressed as $t_h = u_h \times v_h$ where u_h and v_h are cubes in variables $x_1, \ldots, x_m$ and $x_{m+1}, \ldots, x_n$, respectively. Assume that

$u_h = P_1 \times \ldots \times P_m$ for $h = 1,\ldots,e$ and

$u_h \neq P_1 \times \ldots \times P_m$ for $h = e+1,\ldots,k$.

Then C is a tautology iff the subcover $\{t_1,\ldots,t_e\}$ is a tautology.

Proof: The if part is apparent.

($\Rightarrow$) Assume that C is a tautology.

To the contrary, suppose that $\{t_1,\ldots,t_e\}$ is not a tautology. Then there exists a minterm (a,b) such that $(a,b) \notin t_h$ for any $h = 1,\ldots,e$ where a and b are minterms in variables $x_1,\ldots,x_m$ and $x_{m+1},\ldots,x_n$, respectively. Since $t_h = u_h \times v_h$ and $u_h = P_1 \times \ldots \times P_m$ for $h = 1,\ldots,e$, $b \notin v_h$ for any $h = 1,\ldots,e$. On the other hand, since C is unate in variables $x_1,\ldots,x_m$, the cover $\{u_{e+1},\ldots,u_k\}$ is unate. From Lemma 13, $\{u_{e+1},\ldots, u_k\}$ is not a tautology. Thus there exists a minterm c in variables $x_1,\ldots,x_m$ such that $c \notin u_h$ for any $h = e+1,\ldots,k$. Consider the minterm (c,b) in variables $x_1,\ldots,x_n$. Then $(c,b) \notin t_h$ for $h = 1,\ldots,e$ since $b \notin v_h$, and $(c,b) \notin t_h$ for $h = e+1,\ldots,k$ since $c \notin u_h$. We have shown that (c,b) is not covered by any cube in C contradicting that C is a tautology. Thus $\{t_1,\ldots,t_e\}$ must be a tautology.

Example 6 Let $C = \{t_1, t_2, t_3, t_4, t_5\}$ be a cover for a Boolean function defined on $\{0,1,2\} \times \{0,1,2\} \times \{0,1,2\}$ where

$$t_1 = \{0,1,2\} \times \{0,1,2\} \times \{1\}$$
$$t_2 = \{0,1,2\} \times \{0,2\} \times \{2\}$$
$$t_3 = \{0,1,2\} \times \{0\} \times \{1,2\}$$
$$t_4 = \{0\} \times \{0,1,2\} \times \{0,1\}$$
$$t_5 = \{1\} \times \{1,2\} \times \{0,1,2\}.$$

Then C is unate in variable x_1. From Theorem 3, C is a tautology iff the subcover $C' = \{t_1, t_2, t_3\}$ is a tautology. But C' is unate in all 3 variables. From Lemma 13, C' is not a tautology. Thus C is not a tautology.

6. Discussion

We have presented a procedure PRE_EXPAND which can generate all essential primes without generating a prime cover. This procedure can serve as a preprocess for the EXPAND procedure and generate part of a minimum expansion of a given cover. Using the positional cube representation [3] [9], PRE_EXPAND can be implemented to run in time $O(kk' + k^2)$ where k is the number of cubes in the given cover for the Boolean function and k' is the number of cubes in an OFF-cover.

Compared with the EXPAND procedure, PRE_EXPAND is considerably simpler and can run much faster. Moreover, since the EXPAND procedure needs to compute the overexpanded cubes anyway [1] [3] [4], the overhead introduced by PRE_EXPAND is very little. In fact, PRE_EXPAND usually can generate a large set of prime implicants and thus cut down the run time of EXPAND. Consequently, the overall efficiency is improved. The introduction of PRE_EXPAND does not pay off only when the Boolean function has very few partially essential primes.

We have proved a necessary and sufficient condition for detecting essential primes for Boolean functions with multiple-valued inputs. This condition can be checked by using a tautology checking algorithm. Note that this condition uses the asymmetric consensus operation

$$\mathrm{acons}(t_1,t_2) = \bigcup_{\substack{i=1,n \\ S_i \cup R_i \neq R_i}} (S_1 \cap R_1) \times \ldots \times (S_i \cup R_i) \times \ldots \times (S_n \cap R_n)$$

which rarely generates an excessive number of cubes due to the constraint $S_i \cup R_i \neq R_i$ (e.g. if $t_1 \subseteq t_2$, then $\mathrm{acons}(t_1,t_2) = \phi$).

On the other hand, the tautology check is performed only for partially essential primes which constitute a subset of a prime cover. Thus no excessive tautology checking needs to be performed.

The detection of essential primes can be performed right after PRE_EXPAND has generated the partially essential primes. But in practice, it is more appropriate to detect essential primes after EXPAND has generated a prime cover since in this approach the asymmetric consensus may generate fewer cubes.

The concept of unateness has been generalized to Boolean functions with multiple-valued inputs. A cover is unate in a variable if there is a value in the domain of the variable which is not present in the cover. This definition of unateness is more general than that in [4]. But we can still derive a quick check for tautology on a unate cover. Thus the tautology checking algorithm can be improved in speed.

REFERENCES

[1] R.K. Brayton, G.D. Hachtel, C.T. McMullen and A.L. Sangiovanni-Vincentelli, Logic minimization algorithms for VLSI synthesis, Kluwer Academic Publishers, Boston, 1984.

[2] H. Fleisher and L.I. Maissel, "An introduction to array logic", IBM J. Res. Develop., Vol.19, March 1975.

[3] S.J. Hong, R.G. Cain and D.L. Ostapko, "MINI: a heuristic approach for logic minimization", IBM J. Res. Develop., Vol. 18, Sept. 1974.

[4] R.L. Rudell and A.L. Sangiovanni-Vincentelli, "ESPRESSO-MV: algorithms for multiple-valued logic minimization", Proc. of the IEEE Custom Integrated Circuits Conf., 1985.

[5] T. Sasao, "Multiple-valued decomposition of generalized Boolean functions and the complexity of programmable logic arrays", IEEE Trans. on Computers, Vol. C-30, No. 9, Sept. 1981.

[6] T. Sasao, "Tautology checking algorithms for multiple-valued input binary functions and their application", Proc. of the 14th Int. Symp. on Multiple-valued Logic, 1984.

[7] T. Sasao, "Input variable assignment and output phase optimization of PLA's", IEEE Trans. on Computers, Vol. C-33, No. 10, Oct. 1984.

[8] T. Sasao, "An algorithm to derive the complement of a binary function with multiple-valued inputs", IEEE Trans. on Computers, Vol. C-34, No. 2, Feb. 1985.

[9] S.Y.H. Su and P.T. Cheung, "Computer minimization of multivalued switching functions", IEEE Trans. on Computers, Vol. C-21, No. 9, Sept. 1972.

	0	1
0		1
1		

	0	1
0		1,2
1		2

	0	1
0		1,2
1		2

Fig. 1. Consensus Operations

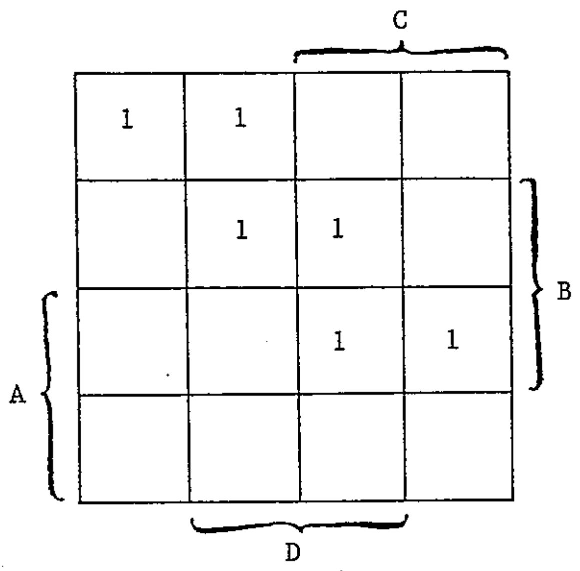

Fig. 2. Generating essential primes before expansion

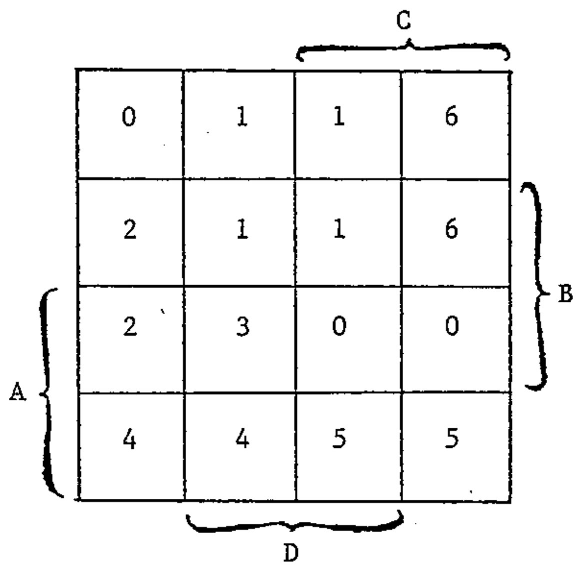

Fig. 3. The PRE_EXPAND procedure

	00	01	11	10
0	2			
1	2			

	00	01	11	10
0	2,1	1,3	3	
1	2			

	00	01	11	10
0	1	1,3	3	
1				

Fig. 4. Detecting essential primes