

## Chapter 1

# TWO-LEVEL LOGIC MINIMIZATION

**Olivier Coudert**

**Tsutomu Sasao**

**Abstract** This chapter presents both exact and heuristic two-level logic minimization algorithms. For exact logic minimization, it shows various techniques to reduce the complexity of covering problems, discusses branching heuristics, and presents several methods to prune the recursions. For heuristic minimization, it presents the core procedures of the ESPRESSO minimizer. Finally, the chapter surveys various works related to two-level logic minimization.

### 1.1 INTRODUCTION

Two-level logic minimization is a fundamental problem in logic synthesis. This chapter consists of two parts. The first parts treat exact logic minimization, and the second part treats heuristic minimization.

The Quine-McCluskey procedure is the classic textbook method used to derive exact minimum two-level logic circuits. However, it is limited to functions up to about 15 variables. The first part of this chapter presents recent progresses on exact two-level logic minimization, with an emphasize on implicit reduction and efficient covering method. With these new methods, exact solutions for functions with many inputs can be obtained in a time comparable to heuristic logic minimizers.

The second part introduces basic operations used in heuristic logic minimization programs.

Let  $f$  be a function from  $\{0, 1\}^n$  into  $\{0, 1, *\}$ . The care set of  $f$  is defined as  $f^{-1}(1)$ , and will be noted  $f^1$ . The don't-care set of  $f$ ,  $f^{-1}(*)$ , will be noted  $f^*$ . It is the set on which the Boolean function  $f$  is not defined. We note  $f^{1*}$  the union of the care set and the don't care set. If the don't care set is empty, the Boolean function is *completely specified* or *total*, else is *incompletely specified*. We will make no distinction between a completely specified Boolean function

and its on-set, since  $f$  is uniquely represented by its on-set. Let  $g$  and  $f$  be Boolean functions. We say that  $g$  covers  $f$  iff  $f^1 \subseteq g^1 \subseteq f^{1*}$ .

A sum-of-products (SOP), or disjunctive normal form, represents a total Boolean function. For instance,  $x_1\bar{x}_2x_3 + x_1x_3x_4 + \bar{x}_2\bar{x}_4$  is a SOP. Two-level logic minimization consists of finding a minimum SOP that covers a given Boolean function  $f$ . Minimization of multi-output Boolean functions (i.e., function from  $\{0, 1\}$  into  $\{0, 1, *\}^m$ ) can be reduced to single-output Boolean function minimization [70, 3, 24, 15].

Two-level logic minimization arises often in logic synthesis, where trying to represent Boolean functions with a two-level NOT, AND and OR netlist [35, 8, 67]. It has various application in reliability analysis [33, 17] and automated reasoning [28, 40, 41, 61, 62]. Section 1.2 presents some aspects of exact minimization. Section 1.3 discusses heuristic minimization.

## 1.2 EXACT LOGIC MINIMIZATION

A product  $p$  is an *implicant* of  $f$  iff  $p \subseteq f^{1*}$ . It is a *prime implicant* (PI) iff there is no other implicant of  $f$  that covers  $p$ . In other words, the set of PIs of  $f$ , which we will note  $PI(f)$ , is the set of maximal implicants w.r.t.  $\subseteq$ . A PI that is the only one to cover some element of  $f^1$  is an *essential prime implicant* (EPI). A SOP  $P$  is *irredundant* iff there does not exist any proper subset of  $P$  that covers  $f$ . A prime and irredundant cover  $P$  of  $f$  is locally minimum (i.e., minimal) in the sense that if we eliminate a literal or a product from  $P$  we no longer have a cover of  $f$ . Conversely any minimum SOP of  $f$  is irredundant and prime, and contains all the EPIs. Finding an irredundant prime cover is a good minimization heuristics [8, 9, 16, 54, 69].

SOP minimization can be viewed as a set covering problem. Let  $Z$  be a set,  $X$  a subset of  $Z$ , and  $Y$  a subset of  $2^Z$ . We say that  $y$  covers  $x$  when  $x \in y$ . Let  $Cost$  be a cost function that applies on subsets of  $Y$ . The set covering problem  $\langle X, Y \rangle$  consists of finding a minimum cost subset  $S$  of  $Y$  such that any  $x$  of  $X$  is covered by some element  $y$  of  $S$ , i.e.,  $X \subseteq \bigcup_{y \in S} y$ .

It is convenient to illustrate a set covering problem  $\langle X, Y \rangle$  with a covering matrix. Its rows (respectively columns) are labeled with elements of  $X$  (respectively  $Y$ ), and an element  $[x, y]$  of the matrix is equal to 1 iff  $x \in y$ . Solving the set covering problem consists in finding a minimum cost subset of columns that cover all the rows. For the sake of simplicity, we will assume in the sequel that the cost of a column is 1.

Fig. 1.1 shows the covering matrix associated with the two-level minimization of  $x_2\bar{x}_4 + x_2\bar{x}_3 + x_1x_4 + \bar{x}_2x_3x_4$ . It has five PIs, as shown in the columns. EPIs are the columns covering the unique “1” occurring in some row. For instance,  $x_2\bar{x}_3$  is essential because it is the only prime that covers the minterm

	$x_2\bar{x}_4$	$x_1x_2$	$x_2\bar{x}_3$	$x_1x_4$	$\bar{x}_2x_3x_4$
0100	1		1		
1100	1	1	1		
0101			1		
1101		1	1	1	
1001				1	
0011					1
1111		1		1	
1011				1	1
0110	1				
1110	1	1			

Figure 1.1. A covering matrix.

“0110”. Also  $x_1x_2$  is the only PI that is not essential, since all elements it covers are also covered by some other primes.

The two-level logic minimization of a Boolean function  $f$  consists in solving the set covering problem  $\langle f^1, PI(f^{1*}) \rangle$ . The principle, often called the Quine–McCluskey procedure [60, 46], is as follows:

- (a) Compute  $PI(f^{1*})$ , the set of all the prime implicants of  $f$ .
- (b) Consider the covering matrix  $\langle f^1, PI(f^{1*}) \rangle$ , with rows labeled by the minterms of  $f$  and columns labeled by the PIs of  $f$ .
- (c) Using reduction techniques and branch-and-bound, solve the covering problem.

Task (a) has an exponential complexity because of the number of prime implicants: it is at most  $O(3^n/\sqrt{n})$  and can be at least  $\Omega(3^n/n)$  [13]. There is no connection between the minimized SOP and the number of prime implicants: there exists a function that has a minimum SOP with  $n$  PIs, but  $2^n - 1$  PIs [50]. Task (b) is also exponential, since all minterms of  $f$  may have to be considered. Task (c) is NP-complete. We address these different tasks in the following section.

### 1.2.1 PRIME IMPLICANT COMPUTATION

Most of the techniques [3, 37, 63, 64, 6, 46, 60, 76, 77, 80] used to compute explicitly all prime implicants of a Boolean function are incremental improvements over Quine’s algorithm [58]. Typically, a technique proceeds by picking a minterm of  $f$ , considering it as a product, and of trying to remove as many as possible literals from it while preserving the implication of  $f$  by the product. Some enhancements compute essential prime implicants [66]. See Section 1.3 for some of these techniques.

The methods cited above iteratively compute one prime implicant at a time, which obviously limit their applications to Boolean functions with few prime implicants (20,000 or less). A far more powerful procedure is described in [14]. It uses a BDD/ZDD-based algorithm to *implicitly* compute all prime implicants, and its complexity is not related to the number of prime implicants [17]. It can handle functions that are out of reach of explicit techniques, and is virtually the only method to address functions with a very large set of prime implicants.

## 1.2.2 SET COVERING REDUCTION

This section presents methods to reduce the complexity of covering problem.

### 1.2.2.1 PARTITIONING

If the rows and the columns of a covering matrix can be permuted to yield a covering matrix that is partitioned in diagonal blocks  $B_k$  as follows, where 1's occur only in these blocks, then any minimum solution of the original problem is the union of minimum solutions of the blocks  $B_k$ . The partitioning of a set covering matrix is easily obtained by noting that two rows that cover a common column are in the same block.

$B_1$			
	$B_2$		
		$\ddots$	
			$B_n$

### 1.2.2.2 ESSENTIALITY

An element  $y$  of  $Y$  is *essential* iff it is the only one that covers an element  $x$  of  $X$ . Since essential columns belong necessarily to any minimum solution, they can be removed from the covering matrix as well as the rows they cover [59].

For instance consider the left-hand side covering matrix of Fig. 1.2. The only element that covers  $x_6$  is  $y_5$ , so  $y_5$  is essential. Column  $y_5$  can be removed from the matrix, as well as all rows it covers, i.e.,  $x_4$  and  $x_6$ . This generates the reduced set covering problem whose matrix is shown on the right-hand side of Fig. 1.2.

### 1.2.2.3 DOMINANCE RELATIONS

An element  $x'$  of  $X$  *dominates*  $x$  iff all elements of  $Y$  that cover  $x'$  also cover  $x$ . This means that once  $x'$  is covered, we no longer care about covering  $x$ . Thus the row labeled with  $x$  can be removed from the matrix without affecting the set of minimum solutions.

	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$
$x_1$	1	1	1	1	
$x_2$	1	1	1	1	
$x_3$	1			1	
$x_4$	1			1	1
$x_5$	1			1	
$x_6$					1

→

	$y_1$	$y_2$	$y_3$	$y_4$
$x_1$	1	1	1	1
$x_2$	1	1	1	1
$x_3$	1			1
$x_5$	1			1

Figure 1.2. Reduction by essentiality.

	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$
$x_1$	1	1	1	1	
$x_2$	1	1	1	1	
$x_3$	1			1	
$x_4$	1			1	1
$x_5$	1			1	
$x_6$					1

→

	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$
$x_3$	1			1	
$x_6$					1

Figure 1.3. Reduction by dominance on  $X$ .

	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$
$x_1$	1	1	1	1	
$x_2$	1	1	1	1	
$x_3$	1			1	
$x_4$	1			1	1
$x_5$	1			1	
$x_6$					1

→

	$y_1$	$y_5$
$x_1$	1	
$x_2$	1	
$x_3$	1	
$x_4$	1	1
$x_5$	1	
$x_6$		1

Figure 1.4. Reduction by dominance on  $Y$ .

In the example shown in Fig. 1.3 on the left,  $x_3$  dominates  $x_1$ ,  $x_2$ ,  $x_4$  and  $x_5$ , and  $x_6$  dominates  $x_4$ . This covering matrix can be then reduced as showed in Fig. 1.3. After this reduction,  $y_2$  and  $y_3$  are trivially useless for solving the set covering problem.

An element  $y'$  of  $Y$  dominates  $y$  iff all elements of  $X$  covered by  $y$  are also covered by  $y'$ , and if  $Cost(y') \leq Cost(y)$ . Since  $y'$  covers all elements covered by  $y$  without any cost penalty,  $y$  can be removed from the matrix. Note that  $y$ -dominance removal is weaker than  $x$ -dominance removal in the following sense:  $y$ -dominance removal does not change the *cost* of the minimum solution, but it can *preclude* some minimum solutions (namely those using  $y$ ). In the example shown in Fig. 1.4 on the left,  $y_1$  dominates  $y_2$ ,  $y_3$ , and  $y_4$ .

#### 1.2.2.4 TRANSPOSING FUNCTION

The covering problem obtained by using dominance based removal and essentiality until saturation is called the *cyclic core*. Explicit algorithms check for row and column removal by looking at them one per one. Using a clever data

	$y_1 =$ $\{1, 2, 3, 4, 5\}$	$y_2 =$ $\{1, 2, 3\}$	$y_3 =$ $\{1, 2, 3, 7\}$	$y_4 =$ $\{1, 2, 3, 4, 5, 7\}$	$y_5 =$ $\{4, 6, 7\}$
$x_1 = \{1\}$	1	1	1	1	
$x_2 = \{2\}$	1	1	1	1	
$x_3 = \{3, 4\}$	1			1	
$x_4 = \{4\}$	1			1	1
$x_5 = \{5\}$	1			1	
$x_6 = \{6, 7\}$					1

$$\begin{aligned}
\tau(x_1) &= \tau(x_2) = & y_1 \cap y_2 \cap y_3 \cap y_4 &= \{1, 2, 3\} \\
\tau(x_3) &= \tau(x_5) = & y_1 \cap y_4 &= \{1, 2, 3, 4, 5\} \\
\tau(x_4) &= & y_1 \cap y_4 \cap y_5 &= \{4\} \\
\tau(x_6) &= & y_5 &= \{4, 6, 7\}
\end{aligned}$$

$$\begin{aligned}
\tau(y_1) &= \tau(y_4) = x_1 \cup x_2 \cup x_3 \cup x_4 \cup x_5 = \{1, 2, 3, 4, 5\} \\
\tau(y_2) &= \tau(y_3) = x_1 \cup x_2 = \{1, 2\} \\
\tau(y_5) &= x_4 \cup x_6 = \{4, 6, 7\}
\end{aligned}$$

	$\{1, 2, 3, 4, 5\}$	$\{4, 6, 7\}$
$\{1, 2, 3, 4, 5\}$	1	
$\{4, 6, 7\}$		1

Figure 1.5. Transposing function based reduction.

structure [69] to represent the matrix allows fast removal check and branch-and-bound (Section 1.2.3). However, the size of the matrix severely limits the range of functions that can be handled.

There exists a more abstract and powerful formulation of cyclic core computation based on *transposing functions*. The idea is to map the original problem onto a set covering problem  $\langle X, Y \rangle$ , so that both  $X$  and  $Y$  are subsets of a lattice  $(Z, \sqsubseteq)$ , with the property that  $y$  covers  $x$  iff  $x \sqsubseteq y$ . Also *all* minimum solutions of the original problem can be obtained from the minimum solutions of the reduced problem (i.e., there is not such a thing as  $y$ -dominance removal losing some minimum solutions). This formulation supports efficient BDD/ZDD based algorithms to *implicitly* produce the cyclic core with a complexity *independent* from the actual size of the matrix. A full presentation of the transposition functions based reduction can be found in [20, 21].

Let us illustrate the idea with a simple example. For a set  $Z$ , the structure  $(2^Z, \subseteq)$  is a lattice. Let  $X$  and  $Y$  be subsets of  $2^Z$ , and let us consider the set covering problem  $\langle X, Y \rangle$  where  $y$  covers  $x$  iff  $x \subseteq y$ . Fig. 1.5 shows

an instance of this problem with  $Z = \{1, 2, 3, 4, 5, 6, 7\}$ . Let us define the function  $\tau$  on  $X$  and  $Y$  as follows:

$$\begin{aligned}\tau(x) &= \bigcap_{y \supseteq x} y \\ \tau(y) &= \bigcup_{x \subseteq y} x\end{aligned}$$

Note that  $y'$  dominates  $y$  iff  $\{x \in X \mid x \subseteq y\} \subseteq \{x \in X \mid x \subseteq y'\}$ , which turns out to be equivalent to  $\tau(y) \subseteq \tau(y')$ . Similarly,  $x'$  dominates  $x$  iff  $\{y \in Y \mid x' \subseteq y\} \subseteq \{y \in Y \mid x \subseteq y\}$ , which is equivalent to  $\tau(x) \subseteq \tau(x')$ . Thanks to this nice property of  $\tau$ , dominance removal reduces  $\langle X, Y \rangle$  into

$$\langle \max_{\subseteq} \tau(X), \max_{\subseteq} \tau(Y) \rangle.$$

Moreover it can be shown that after dominance removal, the essential elements are simply  $X \cap Y$  [20].

Fig. 1.5 shows the values of  $\tau$  on the elements of  $X$  and  $Y$ . In this case, the maximum elements of  $\tau(X)$  w.r.t.  $\subseteq$  are  $\{1, 2, 3, 4, 5\}$  and  $\{4, 6, 7\}$ . They are also the maximal elements of  $\tau(Y)$ . At that point  $X \cap Y$  is indeed the set of the two only essential elements of the reduced matrix.

#### 1.2.2.5 GIMPEL'S REDUCTION

Gimpel proposed a reduction that applies when some row  $x$  is only covered by two columns  $y_1$  and  $y_2$  that moreover have the same cost [30]. Let  $\{x, x_1^1, \dots, x_1^n\}$  and  $\{x, x_2^1, \dots, x_2^m\}$  be the set of elements covered by  $y_1$  and  $y_2$  respectively. Gimpel's reduction consists in removing column  $y_1$ , removing the  $n + m + 1$  rows covered by  $y_1$  and  $y_2$ , and adding  $nm$  new rows  $x^{i,j}$  ( $1 \leq i \leq n$  and  $1 \leq j \leq m$ ) such that the row  $x^{i,j}$  is covered by the set of columns

$$\{y \in Y \mid x_1^i \in y \vee x_2^j \in y\} - \{y_1\}.$$

Let  $S$  be a minimum solution of this new problem. Then a minimum solution of the original set covering problem is derived as follows [65]. If  $S$  is such that  $S \cap \{y \in Y \mid x_1^i \in y\} \neq \emptyset$  for  $1 \leq i \leq n$ , then add  $y_2$  to  $S$ . Otherwise add  $y_1$  to  $S$ .

Consider the example shown in Fig. 1.6. The row  $x$  is only covered by two columns that have the same cost, namely  $y_1$  and  $y_2$ . All rows covered by  $y_1$  and  $y_2$  are removed, namely rows  $x$  up to  $x_2^2$ , and columns  $y_1$  is removed from the matrix. Here we have  $n = m = 2$ , so we add  $nm = 4$  new rows as described above. For instance rows  $x^{2,1}$  is covered by columns that cover either  $x_1^2$  or  $x_2^1$  beside  $y_1$ , i.e.,  $y_2, y_3, y_5$ . Gimpel's reduction produces the covering matrix

	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$
$x$	1	1			
$x_1^1$	1				1
$x_1^2$	1		1		
$x_2^1$		1	1		1
$x_2^2$		1		1	
$x_6$				1	1

→

	$y_2$	$y_3$	$y_4$	$y_5$
$x^{1,1}$	1	1		1
$x^{1,2}$	1		1	1
$x^{2,1}$	1	1		1
$x^{2,2}$	1	1	1	
$x_6$			1	1

Figure 1.6. Gimpel's reduction on  $x$ .

shown on the right-hand side of Fig. 1.6. One of its minimum solution is  $S = \{y_3, y_5\}$ , which intersects both the sets  $\{y \in Y \mid x_1^i \in y\}$  for  $1 \leq i \leq 2$ , i.e.,  $\{y_1, y_5\}$  and  $\{y_1, y_3\}$ , thus adding  $y_2$  to  $S$  produces  $\{y_2, y_3, y_5\}$  which is a minimum solution of the original problem. Another minimum solution of the right-hand side problem is  $S = \{y_2, y_5\}$ . The set  $S$  does not intersects the set  $\{y_1, y_3\}$ , and adding  $y_1$  to  $S$  produces  $\{y_1, y_2, y_5\}$  which is still a minimum solution of the original problem.

Gimpel's reduction yields a new set covering problem with one less column but with  $nm - n - m - 1$  more rows. Adding new rows can produce new dominances between rows and columns, which can reduce the covering matrix. However, from the practical point of view, it is better to allow Gimpel's reduction when it guarantees to produce a smaller covering matrix, i.e., when  $nm - n - m - 1 \leq 0$ .

### 1.2.3 BRANCH AND BOUND

When the reduction processes described above (essentiality, and dominance relations on  $X$  and  $Y$ ) are iteratively applied, it eventually produces a covering matrix that can no longer be reduced. This fixpoint is a *cyclic core*. If this cyclic core is empty, the set of all essential elements that have been found during the reduction process constitutes a minimum solution of the original problem.

If the cyclic core is not empty, we choose an element of  $y$  and generates two subproblems, one assuming that  $y$  belongs to the minimum solution, the other one assuming that it does not. These two subproblems are then simplified (i.e., their cyclic cores are computed), and recursively solved. The minimum solution of the original problem is the minimum of both of the minimum solutions of the two subproblems.

If we know a lower bound of the minimum solution of a subproblem yielded at some point of the branching algorithm, we can prune the recursion as soon as the lower bound is greater or equal to the best solution found so far. It is critical to provide an accurate lower bound to terminate useless searches as early as possible. We will discuss several lower bound computation techniques.

### 1.2.3.1 BRANCHING

Heuristics to properly choose an element of  $Y$  as belonging to the minimum solution are discussed in [8, 46, 69]. A natural heuristics consists in choosing an element  $y$  that cover the largest number of elements of  $X$ . However experimental results show that it is not the best heuristics. A much better heuristics proposed in [69] consists in, given a cyclic core  $\langle X, Y \rangle$ , choosing an element  $y$  that maximizes the following function:

$$\sum_{x \in y} \frac{1}{|\{y \in Y \mid x \in y\}| - 1}$$

This function increases with the number of elements that  $y$  cover, but also with the “quality” of the elements it covers. The less an element  $x$  is covered, the harder it is to cover it, the larger will be its contribution to the function. Thus this function favors elements  $y$ 's that cover  $x$ 's covered by few  $y$ 's.

### 1.2.3.2 PRUNING WITH AN INDEPENDENT SET

Let  $\langle X, Y \rangle$  be a set covering problem, and let  $X'$  be a subset of  $X$  such that for any two different elements  $x_1$  and  $x_2$  of  $X'$ , all elements  $y$  that cover  $x_1$  do not cover  $x_2$  and conversely. The set  $X'$ , which we will call an *independent* subset of  $X$ , provides us with the lower bound

$$\sum_{x \in X'} \min_{y \ni x} Cost(y)$$

since this is the minimum cost necessary to cover the elements of  $X'$ .

Though finding an independent subset that maximizes this lower bound is an NP-complete problem, heuristics in practice yields a quite good lower bound, except for examples that have a large number of  $y$ 's compared to the number of  $x$ 's, or for ill-conditioned covering matrix [22].

### 1.2.3.3 PRUNING WITH THE LEFT-HAND SIDE LOWER BOUND

Let  $C = \langle X, Y \rangle$  be a set covering problem. We note  $C_l = \langle X - y, Y - \{y\} \rangle$  the subproblem that assumes that  $y$  belongs to the minimum solution, and  $C_r = \langle X, Y - \{y\} \rangle$  the subproblem that assumes that  $y$  is not part of the minimum solution. Let  $C.min$  be the cost of its minimum solution,  $C.lower$  some lower bound on this cost, and  $C.path$  the cost of the path that yields  $C$ , i.e., the sum of the costs of all  $y$ 's that have been assumed to belong to the minimum solution before reaching  $C$ . We note  $C.upper$  the global upper bound, i.e., the cost of the best global solution found so far.

Obviously  $C.path + C.lower < C.upper$  must be enforced. If it is not satisfied, the search tree rooted at  $C$  is pruned. This is the usual pruning procedure.

```

S ← ∅;
while X ≠ ∅ do {
  y ← arg miny∈Y  $\frac{Cost(y)}{\Gamma(y \cap X)}$ ;
  X ← X - y;
  S ← S ∪ {y};
}
return S;

```

Figure 1.7. Greedy computation of a solution.

In case  $C.lower$  is the lower bound computed thanks to an independent set (Section 1.2.3.2), a stronger result can be derived: if  $C.path + C_l.lower \geq C.upper$ , then both  $C_l$  and  $C_r$  can be pruned, and  $C_l.lower$  is a *strictly* better lower bound for  $C$  [20, 22]. What is interesting is that if the lower bound of  $C_l$  satisfies the given condition, we do not even need to examine  $C_r$ .

#### 1.2.3.4 PRUNING WITH THE LIMIT LOWER BOUND

Let  $C = \langle X, Y \rangle$  be a set covering problem and let  $X'$  be an independent set of  $C$ . Let  $C.lower = \sum_{x \in X'} \min_{y \ni x} Cost(y)$  the lower bound of  $C$  obtained thanks to  $X'$ . Let

$$Y' = \{y \in Y \mid y \cap X' = \emptyset, C.path + C.lower + Cost(y) \geq C.upper\}$$

be the set of  $y$ 's that do not cover any element of  $X'$ , and whose cost added to  $C.path + C.lower$  exceeds the upper bound. Then  $C$  can be reduced to  $\langle X, Y - Y' \rangle$  [20].

When the limit lower bound is reached for some  $y$ 's, reducing  $\langle X, Y \rangle$  to  $\langle X, Y - Y' \rangle$  makes in practice the recursion terminate immediately, i.e., the lower bound of the latter nearly always exceeds the upper bound, or a better solution is found. To illustrate the gain the limit lower bound produces, assume that  $Cost(y) = 1$  for all  $y$ . Then instead of terminating the recursion when the global lower bound (i.e.,  $C.path + C.lower$ ) reaches  $C.upper$ , it is nearly always pruned when the global lower bound reaches  $C.upper - 1$ . This gain of 1 in the depth of the search can produce an exponential reduction of the space search and reduces dramatically the exploration time. In practice this method dramatically reduces the search space. An extension of the idea of limit lower bound, *negative thinking*, is presented in [31].

#### 1.2.3.5 PRUNING WITH A LOG-APPROXIMATION

This pruning technique relies on an inequality relating the cost of the best solution, and the cost of a solution obtained in a greedy way [23].

Let  $\gamma$  be some strictly positive weighting function defined on the set of rows, and let us define

$$\Gamma(y) = \sum_{x \in y} \gamma(x)$$

Fig. 1.7 shows a subquadratic time algorithm that build a solution in a greedy way (note that the solution is not necessary irredundant). Let  $C = \langle X, Y \rangle$  be a set covering problem,  $C.min$  the cost of its minimum cost solution, and  $S$  the greedy solution produces by the algorithm of Fig. 1.7. Then

$$\frac{Cost(S)}{r} \leq C.min \leq Cost(S)$$

where

$$r = \frac{\max_{y \in Y} \Gamma(y)}{\sum_{k=1} 1/k}$$

This log inequality holds for any function  $\gamma$ . Here, since we are looking for the best lower bound, we are interested in minimizing  $r$  and maximizing  $Cost(S)$ . Since  $\gamma(x)$  captures the difficulty of covering  $x$  (the greater, the more difficult), we can *reverse* the criterion that would yield a good upper bound in order to obtain a good lower bound. For instance, we can take  $\gamma(x) = |x|$ .

#### 1.2.3.6 PRUNING WITH A LPR

A covering problem can be expressed as an Integer Linear Programming (ILP) problem. An integer variable is associated with every column  $y$ , and a row is replaced with the sum of the variables that cover it. The problem becomes:

$$\begin{aligned} & \text{minimize } \sum_{y \in Y} Cost(y) * y, \text{ subject to} \\ & \quad y \in \{0, 1\} \text{ for } y \in Y, \\ & \quad \sum_{y \in Y} \delta_{xy} * y \geq 1 \text{ for } x \in X, \\ & \text{with } \delta_{xy} = 1 \text{ if } y \text{ covers } x, \text{ else } 0 \end{aligned}$$

The linear-programming relaxation of this ILP consists of replacing the integer constraints  $y \in \{0, 1\}$  with the inequality  $0 \leq y \leq 1$ , and allowing the variable  $y$  to take any real value. Note that the optimum solution of the linear-programming relaxation problem is a lower bound of the original ILP.

The lower bound obtained using this principle [44] is of very high quality. It is in theory equal or better to the one obtained with a maximal independent set (Section 1.2.3.2), and its efficiency is often better than the limit lower

bound pruning method (based on a maximal independent set, Section 1.2.3.4), or negative thinking [31].

#### 1.2.4 CONCLUSION

Exact two-level logic minimization is a very hard problem, involving non-polynomial and NP-complete subproblems.

Significant improvements on solving the set covering problem have been done on the branch-and-bound procedure, thanks to better lower bound procedures [20, 23, 31, 44].

The generation of the cyclic core expressing the two-level logic minimization problem has been dramatically improved thanks to a reformulation of the problem in terms of *transposing functions* and efficient BDD/ZDD based implicit algorithms. A full presentation can be found in [20, 21].

### 1.3 HEURISTIC LOGIC MINIMIZATION

Although the classical Quine–McCluskey method has been dramatically improved with implicit technique and better lower bound computation, heuristic methods are more suitable for fast simplification of larger problems. In many applications, exact solutions are not necessary, but near minimum solutions are sufficient. Heuristic methods usually produce solutions that are near to the optimum in a relatively short time. Many heuristic logic minimizers have been developed: PRESTO, MINI, MINI2, POP, ESPRESSO-IIC, and ESPRESSO-MV.

ESPRESSO is the most popular heuristic minimizer. Before describing its algorithm, we will explain some basic operations used in heuristic logic minimizers, illustrated with examples and Karnaugh maps [39].

#### 1.3.1 BASIC OPERATIONS

The following is a simple logic minimization algorithm.

**Algorithm 1.3.1** (*Simplification of SOPs*).

**Step 1:** *MERGE the adjacent products.*

**Step 2:** *EXPAND each product into prime.*

**Step 3:** *DELETE redundant products.*

MERGE uses the identity  $x + \bar{x} = 1$ : two products  $xc$  and  $\bar{x}c$  that differ only in one literal  $x$  can be merge into the single product  $c$ .

**Example 1.3.1** *Consider the following SOP:*

$$F_1 = x_1x_2\bar{x}_3 + x_1\bar{x}_2x_3 + x_1\bar{x}_2\bar{x}_3 + \bar{x}_1x_2x_3 + \bar{x}_1x_2\bar{x}_3 + \bar{x}_1\bar{x}_2x_3.$$

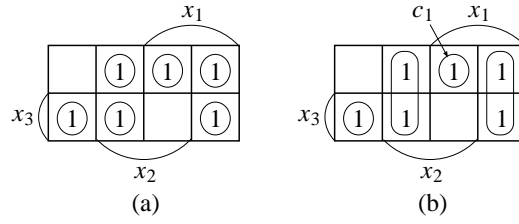


Figure 1.8. MERGE operation.

Fig. 1.8(a) shows the Karnaugh map of this SOP, with its 1st cube  $c_1 = x_1x_2\bar{x}_3$ . The 2nd and 3rd products have different literals in  $x_3$  only, so we can merge them:  $x_1\bar{x}_2x_3 + x_1\bar{x}_2\bar{x}_3 = x_1\bar{x}_2(x_3 + \bar{x}_3) = x_1\bar{x}_2$ . Similarly, the 4th and 5th products have different literals only in  $x_3$ , producing the simplified SOP (Fig. 1.8(b))  $x_1x_2\bar{x}_3 + x_1\bar{x}_2 + \bar{x}_1x_2 + \bar{x}_1\bar{x}_2x_3$

EXPAND looks at each literal  $x$  of a product  $xc$ , and tries to remove it without changing the Boolean function  $f$ . Let us write  $f$  as  $xc + G$ , where  $G$  is a SOP that does not contain the product  $xc$ . If  $\bar{x}c \subseteq f$ , then  $f = \bar{x}c + f = \bar{x}c + (xc + G) = (\bar{x} + x)c + G = c + G$ . So when this condition is met, a literal can be removed from a product. When literals can no longer be removed, the resulting product is prime.

**Example 1.3.2** Consider the following SOP (Fig 1.8.b):

$$F_2 = x_1x_2\bar{x}_3 + x_1\bar{x}_2 + \bar{x}_1x_2 + \bar{x}_1\bar{x}_2x_3.$$

$F_2$  can be written as  $F_2 = c_1 + G$ , with  $c_1 = x_1x_2\bar{x}_3$  and  $G = x_1\bar{x}_2 + \bar{x}_1x_2 + \bar{x}_1\bar{x}_2x_3$ . First, let us check whether we can remove the literal  $x_1$  from  $c_1$ . Let  $c_1 = x_1e_1$  with  $e_1 = x_2\bar{x}_3$ . Let  $c_1(1)$  be the product  $c_1$ , where the literal  $x_1$  is replaced with its complement. Fig. 1.9(a) shows the cube  $c_1(1) = \bar{x}_1e_1 = \bar{x}_1x_2\bar{x}_3$ . Since  $c_1(1) \subseteq F_2$ , we can remove the literal  $x_1$  from  $c_1$ , and have a simpler SOP:

$$F_2 = e_1 + G = x_2\bar{x}_3 + x_1\bar{x}_2 + \bar{x}_1x_2 + \bar{x}_1\bar{x}_2x_3.$$

In this operation, we obtained  $c_2$  by EXPAND'ing the product  $c_1$  into the direction  $\bar{x}_1$  as shown in Fig. 1.9(b).

Next, let us check whether  $c_2 = x_2\bar{x}_3$  can be expanded into the direction  $\bar{x}_2$ . Let  $F_2 = c_2 + G$ , with  $c_2 = x_2e_2$  and  $e_2 = \bar{x}_3$ . Then  $c_2(2) = \bar{x}_2e_2$ . Since,  $c_2(2) \not\subseteq F_2$ , we cannot expand  $c_2$  into the direction  $\bar{x}_2$ . In order to check whether  $c_2$  can be expanded into the direction  $x_3$ , let  $c_2 = \bar{x}_3e_3$  and  $e_3 = x_2$ . Then  $c_2(3) = x_2x_3$ . Since  $c_2(3) \not\subseteq F_2$ ,  $c_2$  cannot be expanded into

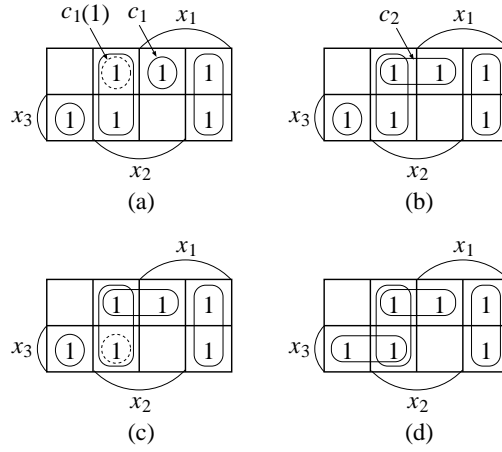


Figure 1.9. EXPAND operation.

the direction  $x_3$  either. Since  $c_2$  cannot be expanded into any direction, it is a prime implicant of  $F_2$ .

Similarly, we can see that  $x_1\bar{x}_2$  and  $\bar{x}_1x_2$  are PIs of  $F_2$ . Finally, by expanding  $\bar{x}_1\bar{x}_2x_3$  into the direction  $x_2$  as shown in Fig. 1.9(c), we obtain the prime implicant  $\bar{x}_1x_3$ . This produces a SOP made of prime implicants only:

$$F_2 = x_2\bar{x}_3 + x_1\bar{x}_2 + \bar{x}_1x_2 + \bar{x}_1x_3,$$

which is shown in Fig. 1.9(d).

Note that the resulting SOP depends on the ordering of the EXPAND operations. In Fig. 1.10(a), if we EXPAND  $c_1$  and  $c_2$  into the direction  $x_2$  and  $\bar{x}_2$  respectively, then we have the ISOP shown in Fig. 1.10(b). As shown in the sequel, the quality of the final solution depends on the direction for expansion. Various heuristics helps ESPRESSO to determine a good ordering.

DELETE eliminates redundant products from an SOP. Let  $c + G$  be a SOP. If  $c \subseteq G$ , then product  $c$  can be removed from the SOP without changing the Boolean function it represents. Once DELETE has deleted all redundant products, the resulting SOP is irredundant.

**Example 1.3.3** Fig. 1.11(a) shows the SOP  $F_3 = x_2\bar{x}_3 + x_1\bar{x}_2 + \bar{x}_1x_2 + \bar{x}_1x_3$ , which consists of prime implicants only.  $F_3$  can be written as

$$\begin{aligned} F_3 &= c_1 + G_1, \text{ where } c_1 = x_2\bar{x}_3 \text{ and} \\ G_1 &= x_1\bar{x}_2 + \bar{x}_1x_2 + \bar{x}_1x_3. \end{aligned}$$

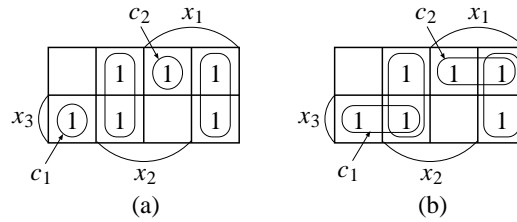


Figure 1.10. EXPAND operation.

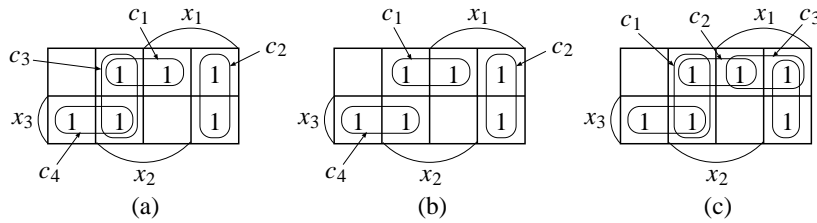


Figure 1.11. DELETE operation.

Since  $c_1 \not\subseteq G_1$ , we cannot delete  $c_1$  from  $F_3$ . Similarly, we cannot delete  $c_2 = x_1\bar{x}_2$ . Let  $c_3 = \bar{x}_1x_2$ , and we have  $G_3 = x_2\bar{x}_3 + x_1\bar{x}_2 + \bar{x}_1x_3$ . Since,  $c_3 \subseteq G_3$ , we can delete  $c_3$ . The last product  $c_4$  cannot be deleted. This yields the ISOP  $F_3 = x_2\bar{x}_3 + x_1\bar{x}_2 + \bar{x}_1x_3$  shown in Fig. 1.11(b).

Note that the quality of the solutions depends on the order of the DELETE operations. In Fig. 1.11(c), if we DELETE  $c_2$  first, then we have an ISOP with four products. On the other hand, if we DELETE  $c_1$  and then  $c_3$ , then we have a minimum SOP with only three products.

### 1.3.2 METHODS TO IMPROVE ISOPS

Algorithm 1.3.1 produces an ISOP, a minimal (i.e., local minimum) solution where neither literal nor a product can be deleted. In many cases, the difference between the numbers of products in a non-minimal ISOP and a minimum solution is not too large. However the difference can be exponential for certain function [75].

Algorithm 1.3.1 can no longer improve the solution shown in Fig. 1.10(b). The operations REDUCE and RESHAPE can be used to produce further improvements.

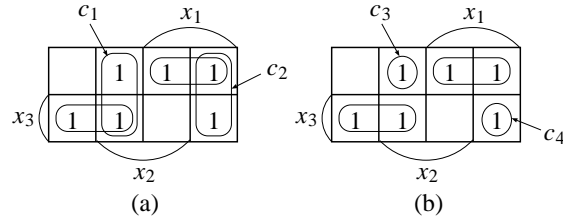


Figure 1.12. REDUCE operation.

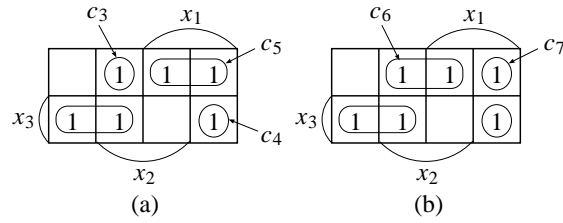


Figure 1.13. RESHAPE operation.

REDUCE is the reverse of EXPAND. REDUCE takes a product  $c$  and reduces the set it represents by adding some literal to  $c$ . When doing so, REDUCE makes sure that it does not change the Boolean function. For example, the REDUCE operation changes  $c_1$  and  $c_2$  in Fig. 1.12(a), into  $c_3$  and  $c_4$  in Fig. 1.12(b), respectively. REDUCE induces new MERGE operations, which can help improving the solution.

RESHAPE replaces a pair of adjacent products into another pair without changing the function represented by the pair. Let  $S$  be a subset of  $B = \{0, 1\}$ . We will use the following notations to represent a two-valued function literal:

$$\begin{aligned}
 x^S &= x && \text{when } S = \{1\}, \\
 &= \bar{x} && \text{when } S = \{0\}, \\
 &= 1 && \text{when } S = \{0, 1\}, \text{ and} \\
 &= 0 && \text{when } S = \phi.
 \end{aligned}$$

Using this notation, RESHAPE can be described as follows. Let  $c_1$  and  $c_2$  be two products:

$$\begin{aligned}
 c_1 &= x_1^{S_1} x_2^{S_2} \dots x_i^{S_i} \dots x_j^{S_j} \dots x_n^{S_n} \\
 c_2 &= x_1^{T_1} x_2^{T_2} \dots x_i^{T_i} \dots x_j^{T_j} \dots x_n^{T_n}
 \end{aligned}$$

They are adjacent iff there exists an  $i$  and  $j$  such that  $S_i \cap T_i = \phi$ ,  $S_j \subseteq T_j$ , and  $S_k = T_k$  for  $k \neq i$ ,  $k \neq j$ . RESHAPE replaces the pair  $(c_1, c_2)$  with the pair  $(c_3, c_4)$  defined as:

$$\begin{aligned} c_3 &= x_1^{S_1} x_2^{S_2} \cdots x_i^{(S_i \cup T_i)} \cdots x_j^{S_j} \cdots x_n^{S_n} \\ c_4 &= x_1^{T_1} x_2^{T_2} \cdots x_i^{T_i} \cdots x_j^{(T_j - S_j)} \cdots x_n^{T_n} \end{aligned}$$

**Example 1.3.4** In Fig. 1.13(a),

$$\begin{aligned} c_3 &= \bar{x}_1 x_2 \bar{x}_3, \text{ and} \\ c_5 &= x_1 1 \bar{x}_3. \end{aligned}$$

Since  $(c_3, c_5)$  satisfies the reshaping conditions, we can replace it by  $(c_6, c_7)$  defined as:

$$\begin{aligned} c_6 &= (\bar{x}_1 + x_1) x_2 \bar{x}_3 = x_2 \bar{x}_3 \\ c_7 &= x_1 \bar{x}_2 \bar{x}_3. \end{aligned}$$

We obtain the SOP shown in Fig. 1.13(b). Note that we can then reduce the number of products by applying the EXPAND operation to  $c_7$ .

### 1.3.3 ESSENTIAL PRIME IMPLICANTS

By applying the EXPAND, the REDUCE, and the RESHAPE operations repeatedly, we have an ISOP. An ISOP has no redundancy, but it may not be a minimum SOP. However, by using EPIs, we can often prove the optimality of an ISOP.

We say that a minterm covered by exactly one prime implicant (which makes this prime an EPI) is a distinguished minterm.

**Example 1.3.5** Let us prove that the ISOP  $\bar{x}_1 \bar{x}_3 \bar{x}_4 + x_1 x_2 + x_3 x_4$  shown in Fig. 1.14 is minimum by showing that it consists only of EPIs. First, consider  $c_1 = \bar{x}_1 \bar{x}_3 \bar{x}_4$ . It covers two minterms  $v_1$  and  $v_2$ . Note that  $c_1$  is the only PI that covers  $v_1 = \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4$ . Thus,  $v_1$  is a distinguished minterm. On the other hand,  $c_1$  and  $x_2 \bar{x}_3 \bar{x}_4$  cover  $v_2 = \bar{x}_1 x_2 \bar{x}_3 \bar{x}_4$ . Thus,  $v_2$  is not a distinguished minterm. Similarly, in  $c_2$ ,  $x_1 x_2 x_3 \bar{x}_4$  and  $x_1 x_2 \bar{x}_3 x_4$  are distinguished minterms. In  $c_3$ ,  $\bar{x}_1 \bar{x}_2 x_3 x_4$ ,  $\bar{x}_1 x_2 x_3 x_4$ , and  $x_1 \bar{x}_2 x_3 x_4$  are distinguished minterms. Since each of  $c_1$ ,  $c_2$ , and  $c_3$  covers distinguished minterm, they are all essential. Thus the ISOP shown in Fig. 1.14 is made of prime implicants that are all essential, which means it is a minimum SOP.

As illustrated in Example 1.3.5, if all the PIs in an ISOP are EPIs, then the ISOP is the unique minimum SOP, and consequently the EXPAND, the

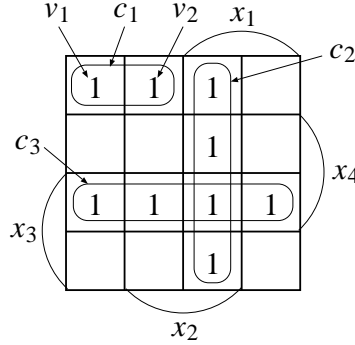


Figure 1.14. Detection of EPIs.

REDUCE, and the RESHAPE iteration can be discontinued. If only a subset  $P$  of the products of an ISOP are essential, we can remove them from the ISOP and add  $P$  to the don't care set. This will produce more simplification and further reduce the SOP.

In the case of control circuits for microprocessors, about a half of the PIs in the minimum SOP are EPIs. Thus, an early detection of the EPIs will reduce the CPU time as well as improve the quality of the solutions. Heuristic minimization algorithms, such as MINI2 and ESPRESSO, detect all the EPIs.

We now describe a method to find the EPIs. In the sequel, a sum of products  $c_1 + c_2 + \dots + c_n$  is represented as the set  $\{c_1, c_2, \dots, c_n\}$ .

**Definition 1.3.1** Given two products

$$\begin{aligned} c_1 &= x_1^{S_1} \cdot x_2^{S_2} \cdot \dots \cdot x_n^{S_n} \text{ and} \\ c_2 &= x_1^{T_1} \cdot x_2^{T_2} \cdot \dots \cdot x_n^{T_n}. \end{aligned}$$

That are distance 1 apart, the **consensus** of  $c_1$  and  $c_2$  is defined as

$$\text{cons}(c_1, c_2) = x_1^{(S_1 \cap T_1)} \cdot x_2^{(S_2 \cap T_2)} \cdot \dots \cdot X_i^{(S_i \cup T_i)} \cdot \dots \cdot X_n^{(S_n \cap T_n)}.$$

We say two products are distance 1 apart if they share no minterms and a minterm of one is adjacent to a minterm of the other. Let  $c$  be a product and  $G$  be a set of products. Then,

$$\text{cons}(c, G) = \bigcup_{c_k \in G} \text{cons}(c, c_k).$$

**Algorithm 1.3.2** (Determine essential prime implicants for two-valued input functions)

1) Let  $F$  be the given set of PIs. Let  $c$  be the PI that we want to determine the essentiality of. Let  $G$  be the set of PIs in  $F$  other than  $c$ , i.e.,  $F = G \cup \{c\}$ .

2) Partition the products in  $G$  into three sets:

$$\begin{aligned} G_1 &= \text{set of products that have common minterms with } c \\ G_2 &= \text{set of products that are adjacent to } c, \text{ that are not in } G_1 \\ G_3 &= \text{set of other products} \end{aligned}$$

3)  $H = \text{cons}(c, G_2)$ .

4)  $c$  is an EPI iff  $c \not\subseteq (H \cup G)$ .

**Example 1.3.6** Find the EPIs in Fig. 1.14. The set of PIs is  $F = \{c_1, c_2, c_3\}$ , where  $c_1 = \bar{x}_1\bar{x}_3\bar{x}_4$ ,  $c_2 = x_1x_2$ , and  $c_3 = x_3x_4$ .

First, let us decide whether  $c_1$  is essential or not.

1) Let  $c = c_1$ ,  $G = \{c_2, c_3\}$ .

2)  $G_1 = \phi$ ,  $G_2 = \{c_2\}$ ,  $G_3 = \{c_3\}$ .

3)  $H = \text{cons}(c_1, c_2) = (\bar{x}_1 + x_1)x_2\bar{x}_3\bar{x}_4 = x_2\bar{x}_3\bar{x}_4$ .

4) Since  $c_1 \not\subseteq (H \cup G_1)$ ,  $c_1$  is an EPI.

Next, let us look at  $c_2$ .

5) Let  $c = c_2$ ,  $G = \{c_1, c_3\}$ .

6)  $G_1 = \{c_3\}$ ,  $G_2 = \{c_1\}$ ,  $G_3 = \phi$ .

7)  $H = \text{cons}(c_2, c_1) = x_2\bar{x}_3\bar{x}_4$ .

8) Since  $c_2 \not\subseteq (H \cup G_1)$ ,  $c_2$  is an EPI.

Then  $c_3$ .

9) Let  $c = c_3$ ,  $G = \{c_2, c_3\}$ .

10)  $G_1 = \{c_2\}$ ,  $G_2 = \phi$ ,  $G_3 = \{c_1\}$ .

11)  $H = \text{cons}(c_3, 0) = 0$ .

12) Since  $c_3 \not\subseteq (H \cup G_1)$ ,  $c_3$  is an EPI.

Thus  $c_1$ ,  $c_2$ , and  $c_3$  are EPIs. On the other hand,  $c_4 = x_2\bar{x}_3\bar{x}_4$  is a PI, but not an EPI. This can be verified as follows:

13) Let  $c = c_4$ ,  $G = \{c_1, c_2, c_3\}$ .

$$14) G_1 = \{c_1, c_2\}, G_2 = \phi, G_3 = \{c_3\}.$$

$$15) H = \text{cons}(c_4, 0) = 0.$$

16) Since  $c_4 \subseteq (H \cup G_1)$ ,  $c_4$  is not an EPI.

### 1.3.4 FINDING AN ISOP WITH FEWER PRODUCTS

The IRREDUNDANT procedure generates an ISOP with as few products as possible from the given set of PIs.

#### Algorithm 1.3.3 (IRREDUNDANT)

1 Let  $F = \{c_1, c_2, \dots, c_k\}$  be the given set of PIs. Partition the PIs of  $F$  into three sets:

a) Relatively essential set:  $ER = \{c_i \mid c_i \not\subseteq \sum_{c \in F - \{c_i\}} c\}$ .

b) Totally redundant set:  $RT = \{c_i \mid c_i \subseteq ER\}$ .

c) Partially redundant set:  $RP = F - (ER \cup RT)$ .

2 Find a subset of  $RP$  that together with  $ER$  covers  $F$ . For each  $c_i$  in  $RP$ , find the set  $H(c_i)$  of minimum sets  $MS \subseteq RP$ , such that  $c_i \subseteq (RP - MS) \cup ER$ .

3 Construct the logic function

$$CV(f) = \bigwedge_{c_i \in RP} \left( \bigwedge_{MS \in H(c_i)} \left( \bigvee_{c_j \in MS} g_j \right) \right).$$

4 Convert  $CV(f)$  into an SOP, and find the product with the minimum number of literals.

5 The set of PIs that corresponds to the above product together with PIs in  $ER$  forms the minimal cover.

**Example 1.3.7** Consider the SOP shown in Fig. 1.15.

1)  $F = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7\}$  is a given set of PIs. Partition  $F$  into three sets:  $ER = \{c_1, c_7\}$ .  $RT = \phi$ .  $RP = \{c_2, c_3, c_4, c_5, c_6\}$ .

2)  $H(c_2) = [\{c_2, c_3\}]$ ;  $H(c_3) = [\{c_2, c_3\}, \{c_3, c_4\}]$ ;  $H(c_4) = [\{c_3, c_4\}, \{c_4, c_5\}]$ ;  
 $H(c_5) = [\{c_4, c_5\}, \{c_5, c_6\}]$ ;  $H(c_6) = [\{c_5, c_6\}]$ .

$$\begin{aligned} 3-4) \quad CV(f) &= (g_2 + g_3)(g_2 + g_3)(g_3 + g_4)(g_3 + g_4) \\ &\quad (g_4 + g_5)(g_4 + g_5)(g_5 + g_6)(g_5 + g_6) \\ &= (g_2 + g_3)(g_3 + g_4)(g_4 + g_5)(g_5 + g_6) \\ &= g_2g_4g_5 + g_2g_4g_6 + g_3g_5 + g_3g_4g_6. \end{aligned}$$

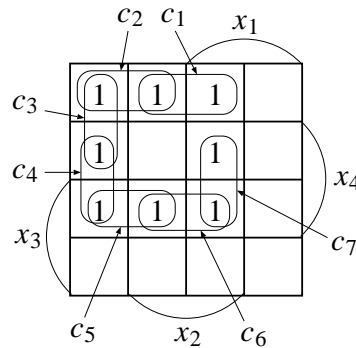


Figure 1.15. Given set of PIs.

- 5)  $g_3g_5$  is the product with the fewest literals, and the corresponding cover is  $\{c_3, c_5\}$ . Thus  $\{c_3, c_5\} \cup ER = \{c_1, c_3, c_5, c_7\}$  is a minimum cover.

Algorithm 1.3.3 and Example 1.3.7 show the principle of the method. The actual procedure in ESPRESSO is more complicated, because steps 3—5 correspond to a minimum covering algorithm. In particular step 4 can be non-polynomial.

Note that  $F$  does not necessarily contain all the PIs. Thus,  $ER$  contains all the EPIs, and possibly non-essential PIs. If  $F$  contains all the PIs, then  $ER$  is equal to the set of all the EPIs for the function, and the IRREDUNDANT operation produces an minimum SOP.

### 1.3.5 ESPRESSO

ESPRESSO is the most popular heuristic two-level logic minimizer. First, it obtains the complement of the original cover, which will be used in the EXPAND operation. Second, it applies the EXPAND and IRREDUNDANT operation to obtain an ISOP. Third, it extracts the set of EPIs, and it iterates the REDUCE, the EXPAND and the IRREDUNDANT operations until no product can be reduced any more. Fourth, it attempts to REDUCE and EXPAND by using different heuristics. Finally, ESPRESSO tries to reduce the number of connections in the output part.

## 1.4 CONCLUSION

Two-level logic minimization, both exact and heuristics, have received much of attentions, since they are critical in logic synthesis, as well as other real-life applications. It is a very mature field, but due to the difficulty of the problem,

is still an active research domain. Beside improving on the existing methods, with the most spectacular improvements being implicit minimization and new efficient lower bound techniques during branch-and-bound, there are attempts at identifying class of functions for which logic minimization can be categorized in the polynomial space.

## History and Related Topics

### Exact Logic Minimization.

Quine stated the two-level minimization problem for a total Boolean function in 1952 [58]. Exact minimization yielded the classic Quine–McCluskey procedure [60, 46], and a number of papers improved on this minimization procedures, focusing on cyclic core computation and branch-and-bound techniques [4, 35, 8, 68, 69, 50, 36, 9, 20, 22, 23].

After the introduction of BDDs [10, 11], *implicit minimization procedures* were proposed to overcome the bottleneck of prime implicant computation and cyclic core computation. A first implicit BDD-based cyclic core computation procedure was proposed in [79]. Finally, a BDD/ZDD-based implicit cyclic core computation using transposing functions was introduced in [18, 20, 21].

ESPRESSO-EXACT [69] and SCHERZO [18] are the most well known, respectively explicit and implicit/explicit, exact minimization procedures. Textbooks that explain exact SOP minimization include [20, 34, 42, 48, 56].

### Heuristic Logic Minimizers.

MINI, a multiple-valued logic minimizer to simplify PLA with decoders, is described in [35]. MINI uses the complement of the given function to expand the implicant efficiently. PRESTO [78] is a simple logic minimizer that uses only EXPAND and REDUCE of the output part. It uses tautology to expand the implicant, which is slower than the MINI and ESPRESSO strategy. PRESTO was used in ABEL, a design system for programmable logic devices.

In the early 1980's, PLAs started to be used in control parts of various 32-bit microprocessors, which triggered a strong interest in SOP minimization. MINI2 [71], an improvement of MINI, detects essential prime implicants (EPIs), and uses divide and conquer strategy for multiple-valued logic. A straightforward method to detect EPIs using the set of all the PIs [27] requires excessive memory. MINI2 detects EPIs without using the set of all the PIs [43, 71].

ESPRESSO-IIC, an improvement of MINI, is described in [8]. It is a two-valued logic minimizer. The important improvements are introduction of unate recursive paradigm and the IRREDUNDANT operation. ESPRESSO-MV, a multi-valued version of ESPRESSO-IIC, is described in [68]. It is widely

used for two-level logic minimization. Implicit heuristic minimizations are described in [16, 19, 53].

#### Various Topics.

WSOP (ISOP that has the maximum number of PIs) [75]; optimization of SOPs for multi-output functions using MVL [3, 47, 35, 70]; special hardware for logic minimization [72, 78]; other logic minimization algorithms [1, 2, 5, 7, 12, 25, 53, 63, 81, 83]; benchmark functions [82]; prime implicant computation [46, 55, 59, 80, 26, 77, 14]; implicit PI computation algorithm [14, 45, 73]; logic functions with many PIs [29]; average numbers of PIs for the functions with a given number of minterms [52]; functions with maximal number of PIs [38]; a class of functions whose SOPs are difficult to minimize [57, 75]; minimization of SOPs by functional decomposition [74]. De Micheli's book [51] and Hachtel–Somenzi's book [32] have excellent chapters on two-level optimization. Muroga's book has a good survey of classic approach [56].

#### References

- [1] Z. Arevalo, J. G. Bredeson, "A method to simplify a Boolean function into a near minimal sum-of-products for programmable logic arrays," *IEEE Trans. on Comput.*, C-27, pp. 1028–1039, Nov. 1978.
- [2] M. Auguin, F. Boeri, and C. Andre, "An algorithm for designing multiple Boolean functions: Application to PLA's," *Digital Process*, Vol. 4, No. 3-4, pp. 215–230, 1978.
- [3] T.C. Bartee, "Computer design of multiple-output logical networks," *IRE Trans. on Elect. and Comp.*, pp. 21–30, March 1961.
- [4] T.C. Bartee, I.L. Lebow, I.S. Reed, *Theory and Design of Digital Machines*, McGraw-Hill, New York, 1962.
- [5] Ph. W. Besslich and P. Pichlbauer, "Fast transform procedure for the generation of near minimal covers of Boolean functions," *IEE Proc.*, Vol. 128, Part E, No. 6, pp. 250–254, Nov. 1981.
- [6] N.N. Biswas, *Introduction to Logic and Switching Theory*, Gordon & Breach Science, New York, 1975.
- [7] N. N. Biswas, "Computer aided minimization procedure for Boolean functions," *DAC'84*, pp. 699–702, June 1984.
- [8] R.K. Brayton, G.D. Hachtel, C.T. McMullen, and A.L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, Dordrecht, 1984.
- [9] R.K. Brayton, P.C. McGeer, J. Sanghavi, and A.L. Sangiovanni-Vincentelli, "A new exact minimizer for two-level logic synthesis," *Logic Synthesis and Optimization*, pp. 1–31, T. Sasao Ed., Kluwer Academic Publishers, Dordrecht, 1993.
- [10] R.E. Bryant, "Graph-based algorithms for Boolean functions manipulation," *IEEE Trans. on Comp.*, C-35, No. 8, pp. 677–692, Aug. 1986.
- [11] R.E. Bryant, "Symbolic Boolean Manipulations with Ordered Binary Decision Diagrams," *ACM Computing Surveys*, Vol. 24, No. 3, pp. 293–318, Sept. 1992.

- [12] G. Caruso, "A selection algorithm for switching function minimization," *IEEE Trans. on Comput.*, C-33, No. 1, pp. 91–97, Jan. 1984.
- [13] A.K. Chandra, G. Markowsky, "On the Number of Prime Implicants," *Discrete Mathematics*, Vol. 24, pp. 7–11, 1978.
- [14] O. Coudert and J.C. Madre, "Implicit and incremental computation of primes and essential primes of Boolean functions," *Proc. 29th DAC*, CA, USA, pp. 36–39, June 1992.
- [15] O. Coudert and J.C. Madre, "A new graph based prime computation technique," *Logic Synthesis and Optimization*, pp. 33–57, T. Sasao Ed., Kluwer Academic Publishers, Dordrecht, 1993.
- [16] O. Coudert and J.C. Madre, "Towards a symbolic logic minimization algorithm," *Proc. VLSI Design*, Bombay, India, Jan. 1993.
- [17] O. Coudert and J.C. Madre, "Fault tree analysis:  $10^{20}$  prime implicants and beyond," *Proc. Annual Reliability and Maintainability Symp.*, Atlanta, GA, USA, pp. 240–245, Jan. 1993.
- [18] O. Coudert, J.C. Madre, and H. Fraise, "A new viewpoint on two-level logic minimization," *Proc. 30th DAC*, Dallas, TX, USA, pp. 625–630, June 1993.
- [19] O. Coudert, J. C. Madre, H. Fraise, and H. Touati, "Implicit prime cover computation: An overview," *Proc. of SASIMI'93*, Nara, Japan, Oct. 1993.
- [20] O. Coudert, "Two-level logic minimization: An overview," *Integration*, Vol. 17, No. 2, pp. 97–140, Oct. 1994.
- [21] O. Coudert, "Doing two-level logic minimization 100 times faster," *Proc. of Symposium on Discrete Algorithms (SODA)*, pp. 112–121, CA, Jan. 1995.
- [22] O. Coudert and J. C. Madre, "New ideas for solving covering problems," *Proc. of 32nd DAC*, pp. 641–646, CA, June 1995.
- [23] O. Coudert, "On Solving Covering Problems," *Proc. of 33rd DAC*, pp. 197–202, Las Vegas NV, June 1996.
- [24] R.B. Cutler and S. Muroga, "Useless prime implicants of incompletely specified multiple-output switching functions," *Int'l Journal of Computer and Information Sciences*, Vol. 9, No. 4, 1980.
- [25] M. R. Dagenais, V. K. Agarwal, and N. C. Rumin, "McBoole: A new procedure for exact logic minimization," *IEEE TCAD*, Vol. CAD-5, No. 1, pp. 229–233, Jan. 1986.
- [26] S. R. Das, "A new algorithm for generating prime implicants," *IEEE Trans. on Comput.*, C-20, No. 12, pp. 1614–1615, 1971.
- [27] D. L. Dietmeyer, *Logic Design of Digital Systems (2nd Edition)*, Allyn and Bacon Inc., Boston, 1978.
- [28] J. Doyle, "A truth maintenance system," *Artificial Intelligence*, Vol. 12, pp. 231–271, 1979.
- [29] B. Dunham and R. Fridshal, "The problem of simplifying logical expressions," *Journal of Symbolic Logic*, Vol. 24, pp. 17–19, 1959.
- [30] J.F. Gimpel, "A reduction technique for prime implicant tables," *IEEE Trans. on Elect. Comp.*, EC-14, pp. 535–541, 1965.
- [31] E.I. Goldberg, L.P. Carloni, T. Villa, R.K. Brayton, and A.L. Sangiovanni-Vincentelli, "Negative thinking by incremental problem solving: Application to unate covering," *Proc. of ICCAD'97*, pp. 91–97, 1997.

- [32] G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*, Kluwer Academic Publishers, Boston, 1996.
- [33] D.F. Hasl, "Advanced concepts in fault tree analysis," *Proc. System Safety Symposium*, Seattle, USA, June 1965.
- [34] F. J. Hill and G. R. Peterson, *Computer Aided Logic Design with Emphases on VLSI*, Wiley, 1993.
- [35] S.J. Hong, R.G. Cain, and D.L. Ostapko, "MINI: A heuristic approach for logic minimization," *IBM Journal R&D*, pp. 443–458, 1974.
- [36] S.J. Hong and S. Muroga, "Absolute minimization of completely specified switching functions," *IEEE Trans. on Comp.*, Vol. 40, pp. 53–65, 1991.
- [37] H.R. Hwa, "A method for generating prime implicants of a Boolean expression," *IEEE Trans. on Comp.*, pp. 637–641, June 1974.
- [38] Y. Igarashi, "An improved lower bound on the maximum number of prime implicants," *IEICE Trans.*, **E62-6**, pp. 389–394, June 1979.
- [39] M. Karnaugh, "The map method for synthesis of combinational logic circuits," *AIEE Trans. on Comm. & Elect.*, Vol. 9, pp. 593–599, 1953.
- [40] J. De Kleer, "An assumption-based TMS," *Artificial Intelligence*, Vol. 28, pp. 127–162, 1986.
- [41] J. De Kleer and B.C. Williams, "Diagnosing multiple faults," *Artificial Intelligence*, Vol. 32, pp. 97–130, 1987.
- [42] Z. Kohavi, *Switching and Finite Automata Theory*, McGraw-Hill Book Co., 1970.
- [43] Y. S. Kuo, "Generating essential primes for a Boolean function with multiple-valued inputs," *IEEE Trans. on Comput.*, Vol. C-36, No. 3, March 1987.
- [44] S. Liao and S. Devadas, "Solving covering problems using LPR-based lower bounds," *Proc. 34th DAC Conference*, Anaheim, CA, USA, pp. 117–120, June 1997.
- [45] B. Lin, O. Coudert, and J.C. Madre, "Symbolic prime generation for multiple-valued functions," *Proc. 29th DAC*, pp. 40–44, June 1992.
- [46] E.L. Jr. McCluskey, "Minimization of Boolean functions," *Bell System Technical Journal*, Vol. 35, pp. 1417–1444, April 1959.
- [47] E.L. Jr. McCluskey and H. Schorr, "Essential multiple output prime implicants," *Proc. Symp. on Math. Theory of Automata*, Vol. 12, Polytech. Inst. of Brooklyn, New York, NY, pp. 437–457, April 1962.
- [48] E. J. McCluskey, *Introduction to the Theory of Switching Circuits*, McGraw-Hill, New York, 1965.
- [49] P.C. McGeer, J. Sanghavi, R.K. Brayton, and A.L. Sangiovanni-Vincentelli, "ESPRESSO-SIGNATURE: A new exact minimizer for logic functions," *IEEE Trans. on VLSI*, Vol. 1, No. 4, pp. 432–440, Dec. 1993.
- [50] C. McMullen and J. Shearer, "Prime implicants, minimum covers, and the complexity of logic simplification," *IEEE Trans. on Comp.*, Vol. C-35, pp. 761–762, Aug. 1986.
- [51] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.
- [52] F. Mileto and G. Putzolu, "Average values of quantities appearing in Boolean function minimization," *IEEE TEC*, Vol. EC-13, No. 4, pp. 87–92, April 1964.
- [53] S. Minato, "Fast generation of prime-irredundant covers from binary decision diagrams," *IEICE Trans. Fundamentals*, Vol. E76-A, No. 6, pp. 967–973, June 1993.

- [54] E. Morreale, "Recursive operators for prime implicant and irredundant normal form determination," *IEEE Trans. on Comp.*, Vol. C-19, PP. 504–509, June 1970.
- [55] T. H. Mott Jr., "Determination of irredundant formal forms of a truth function by iterated consensus of the prime implicants," *IRE TEC*, pp. 245–252, June 1960.
- [56] S. Muroga, *Logic design and Switching Theory*, Wiley-Interscience Publication, 1979.
- [57] D. L. Ostapko and S. J. Hong, "Generating test examples for heuristic Boolean minimization," *IBM J. Res. and Develop.*, Vol. 18, pp. 459–464, Sept. 1974.
- [58] W.V.O. Quine, "The problem of simplifying truth functions," *American Math. Monthly*, Vol. 59, pp. 521–531, 1952.
- [59] W.V.O. Quine, "A way to simplify truth functions," *American Math. Monthly*, Vol. 62, pp. 627–631, 1955.
- [60] W.V.O. Quine, "On cores and prime implicants of truth functions," *American Math. Monthly*, Vol. 66, pp. 755–760, 1959.
- [61] R. Reiter, "A theory of diagnosis from first principles," *Artificial Intelligence*, Vol. 32, pp. 57–95, 1987.
- [62] R. Reiter and J. de Kleer, "Foundations for assumption-based truth maintenance systems," *Proc. AAAI National Conference '87*, Seattle, pp. 183–188, July 1987.
- [63] V.T. Rhyne, P.S. Noe, M.H. McKinney, and U.W. Pooch, "A new technique for the fast minimization of switching functions," *IEEE Trans. on Comp.*, Vol. C-26, No. 8, pp. 757–764, 1977.
- [64] J.A. Robinson, "A machine-oriented logic based on the resolution principle," *Journal of ACM*, Vol. 12, pp. 23–41, 1965.
- [65] S. Robinson and R. House, "Gimpel's reduction technique extended to the covering problem with costs," *IEEE Trans. on Elect. Comp.*, Vol. EC-16, pp. 509–514, Aug. 1967.
- [66] J.P. Roth, "Algebraic Topological Methods for the Synthesis of Switching Systems," *Trans. of American Math. Society*, Vol. 88, No. 2, pp. 301–326, 1958.
- [67] R.L. Rudell, *Multiple-Valued Logic Minimization for PLA Synthesis*, Research Report, UCB M86/65, 1986.
- [68] R.L. Rudell and A.L. Sangiovanni-Vincentelli, "Multiple valued minimization for PLA optimization," *IEEE Trans. on CAD*, Vol. 6, No. 5, pp. 727–750, Sept. 1987.
- [69] R.L. Rudell, *Logic Synthesis for VLSI Design*, PhD Thesis, UCB/ERL M89/49, 1989.
- [70] T. Sasao, "An application of multiple-valued logic to a design of programmable logic arrays," *Proc. Int'l Symp. on Multiple-Valued Logic*, pp. 65–72, May 1978.
- [71] T. Sasao, "Input variable assignment and output phase optimization of PLA's," *IEEE TC*, Vol. C-33, No. 10, pp. 879–894 Oct. 1984.
- [72] T. Sasao, "HART: A hardware for logic minimization and verification," *ICCD'85*, New York, pp. 713–718, Oct. 7–10, 1985.
- [73] T. Sasao, "Ternary decision diagrams and their applications," Chap. 12 of *Representations of Discrete Functions*, Kluwer Academic Publishers, 1996.
- [74] T. Sasao and J. T. Butler, "On the minimization of SOPs for bi-decomposable functions," *ASP-DAC'2001*, Japan, Jan. 2001.
- [75] T. Sasao and J. T. Butler, "Worst and best irredundant sum-of-products expressions," *IEEE Trans. on Comp.*, Vol. 50, No. 9, Sept. 2001.

- [76] J.R. Slagc, C.L. Chang, and R.C.T. Lee, "Completeness theorems for semantics resolution in consequence finding," *Proc. Int. Join Conference on Artificial Intelligence*, pp. 281–285, 1969.
- [77] J.R. Slagc, C.L. Chang, and R.C.T. Lee, "A new algorithm for generating prime implicants," *IEEE Trans. on Comp.*, Vol. C-19, No. 4, pp. 304–310, 1970.
- [78] A. Svoboda and D. E. White, *Advanced Logical Circuit Design Techniques*, Garland Press, New York, 1979.
- [79] G.M. Swamy, P. McGeer, and R.K. Brayton, "A fully Quine–McCluskey procedure using BDD's," *Proc. IWLS'93*, May 1993.
- [80] P. Tison, "Generalized consensus theory and application to the minimization of Boolean functions," *IEEE Trans. on Elect. Comp.*, Vol. EC-16, No. 4, pp. 446–456, 1967.
- [81] K-N. Wong, "New heuristic for the exact minimization of logic functions," *IEEE Int'l Symp. on Circuits and Systems*, pp. 1865–1868, 1988.
- [82] S. Yang, *Logic Synthesis and Optimization Benchmarks User Guide*, Microelectronics Center of North Carolina, Jan. 1991.
- [83] M. H. Young and S. Muroga, "Symmetric minimal covering problem and minimal PLA's with symmetric variables," *IEEE Trans. on Comput.*, Vol. C-34, No. 6, pp. 523–541, June 1985.