EXACT ALGORITHMS FOR OUTPUT ENCODING, STATE ASSIGNMENT AND FOUR-LEVEL BOOLEAN MINIMIZATION

by

Srinivas Devadas and A. Richard Newton

Memorandum No. UCB/ERL M89/8

3 February 1989

COVER PAGE

EXACT ALGORITHMS FOR OUTPUT ENCODING, STATE ASSIGNMENT AND FOUR-LEVEL BOOLEAN MINIMIZATION

by

Srinivas Devadas and A. Richard Newton

Memorandum No. UCB/ERL M89/8

3 February 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

10	inp1	1010
01	inp1	0110
10	inp2	1010
-1	inp2	1011
1-	inp3	0110
0-	inp3	1001
--	inp4	0010
--	inp5	1101

(a)

0001	out1
00-0	out2
0011	out2
0100	out3
1000	out3
1011	out4
1111	out5

(b)

Figure 1: Symbolic Covers

1 Introduction

Encoding problems in switching theory include the input encoding and output encoding problems which involve the assignment of binary codes to symbolic inputs and outputs so as to minimize a given cost function (typically, the area of the resulting logic network). State assignment, one of the oldest problems in automata theory, is also an encoding problem. If we view the State Transition Table (STT) of a finite state machine (FSM) as a truth-table with a symbolic input corresponding to the present states and a symbolic output corresponding to the next states, then state assignment can be seen as an input-output encoding problem for the truth-table. This encoding problem has associated constraints on the equality of codes that are assigned to the input and output symbols (symbols that correspond to the same symbolic state in the sequential machine). Depending on the targeted implementation, the goal of the encoding step, be it input or output encoding or state assignment, varies.

Encoding problems are difficult because they typically have to model a complicated optimization step that follows. For instance, if we have symbolic truth-tables like those in Figure 1, which are to be implemented in PLA form, one wishes to code the *inp*1, .., *inpN* (*out*1, .., *outM*) so as to minimize the number of product terms (or the area) of the resulting PLA *after* two-level Boolean minimization. A straightforward, exhaustive search technique to find a optimum encoding would require $O(N!)$ (($O(M!)$)) *exact* two-level Boolean minimizations. Two-level Boolean minimization algorithms are very well developed – the programs ESPRESSO-EXACT [15] and McBOOLE [4] minimize large functions exactly within reasonable amounts of CPU time. However, the *number* of required minimizations makes an exhaustive search approach to optimum encoding infeasible for anything but the smallest problems.

Exact Algorithms for Output Encoding, State Assignment and Four-Level Boolean Minimization

Srinivas Devadas
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology, Cambridge

A. Richard Newton
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

Abstract

In this paper, we present efficient, exact algorithms for the problems of output encoding, state assignment and four-level Boolean minimization. All previous automatic approaches to encoding problems have involved the use of heuristic techniques. Other than the straightforward, exhaustive search procedure, no exact solution methods have been proposed.

The problems of output encoding and state assignment targeting two-level logic implementations involve finding appropriate binary codes for the symbolic outputs or states so a minimum number of product terms results after two-level Boolean minimization. A straightforward, exhaustive search procedure requires $O(N!)$ exact Boolean minimizations, where N is the number of symbolic outputs or states. We propose a novel minimization procedure of *prime implicant generation and covering that operates on symbolic outputs*, rather than binary-valued outputs, for solving the output encoding problem. An exact solution to this minimization problem is also an exact solution to the encoding problem. While our covering problem is more complex than the classic unate covering problem, a single logic minimization step replaces $O(N!)$ minimizations.

The input encoding problem can be exactly solved using multiple-valued Boolean minimization. We present an exact algorithm for state assignment by generalizing our output encoding approach to the multiple-valued input case.

Four-level Boolean minimization entails finding a cascaded pair of two-level logic functions that implement another logic function, such that the sum of the product terms in the two cascaded functions or truth-tables is minimum. Four-level Boolean minimization can be formulated as an encoding problem and solved exactly using our algorithms.

We present preliminary experimental results which indicate that medium-sized problems can be solved exactly. Computationally efficient heuristic approaches based on the exact algorithms are proposed for output encoding, state assignment and four-level Boolean minimization.

10	inp1	1010
01	inp1	0110
10	inp2	1010
-1	inp2	1011
1-	inp3	0110
0-	inp3	1001
--	inp4	0010
--	inp5	1101

(a)

0001	out1
00-0	out2
0011	out2
0100	out3
1000	out3
1011	out4
1111	out5

(b)

Figure 1: Symbolic Covers

1 Introduction

Encoding problems in switching theory include the input encoding and output encoding problems which involve the assignment of binary codes to symbolic inputs and outputs so as to minimize a given cost function (typically, the area of the resulting logic network). State assignment, one of the oldest problems in automata theory, is also an encoding problem. If we view the State Transition Table (STT) of a finite state machine (FSM) as a truth-table with a symbolic input corresponding to the present states and a symbolic output corresponding to the next states, then state assignment can be seen as an input-output encoding problem for the truth-table. This encoding problem has associated constraints on the equality of codes that are assigned to the input and output symbols (symbols that correspond to the same symbolic state in the sequential machine). Depending on the targeted implementation, the goal of the encoding step, be it input or output encoding or state assignment, varies.

Encoding problems are difficult because they typically have to model a complicated optimization step that follows. For instance, if we have symbolic truth-tables like those in Figure 1, which are to be implemented in PLA form, one wishes to code the *inp*1, .., *inpN* (*out*1, .., *outM*) so as to minimize the number of product terms (or the area) of the resulting PLA *after* two-level Boolean minimization. A straightforward, exhaustive search technique to find a optimum encoding would require $O(N!)$ (($O(M!)$) *exact* two-level Boolean minimizations. Two-level Boolean minimization algorithms are very well developed – the programs ESPRESSO-EXACT [15] and McBOOLE [4] minimize large functions exactly within reasonable amounts of CPU time. However, the *number* of required minimizations makes an exhaustive search approach to optimum encoding infeasible for anything but the smallest problems.

Early approaches to state assignment (e.g. [1], [9], [7], [20]) targeted sum-of-products implementations of finite state machines. Heuristic search methods attempted to produce a state encoding that minimized the number of product terms in the resulting truth-table or the number of gates in the resulting two-level network. More recently, multi-level combinational logic implementations have been targeted [5] [21].

In [14], it was shown that the input encoding problem, when the objective is to minimize the number of product terms in the eventual PLA, can be solved exactly by means of an exact multiple-valued Boolean minimization. A minimum cardinality cover equal to the cardinality of a one-hot coded cover is obtained and the number of bits required to code the symbolic inputs can be minimized as a secondary objective. The program KISS [14] approximates the state assignment algorithm as one of input encoding and produces FSMs implemented as PLAs whose product term cardinality is no greater than that of a one-hot coded FSM. However, since the next state space is completely ignored, no guarantees as to the global optimality of the result for the original state assignment problem can be made.

Optimum state assignment requires the optimum integration of input and output encoding algorithms. Unfortunately, no exact methods for output encoding (other than the trivial exhaustive search method) have been proposed to date. Heuristic output encoding strategies (e.g. [13], [17]) have been proposed and used in conjunction with the approach in [14] to state assignment.

The problem of minimizing a cascaded chain of linked PLAs, multi-level Boolean minimization, is one of great theoretical and practical interest. Several heuristic methods involving algebraic and Boolean decomposition techniques have been proposed (e.g. [2], [8], [6]). An exact factorization algorithm for single-output functions was proposed in [11]. In [6], the problem of decomposing a given two-level function into a cascaded pair of two-level functions was posed as an encoding problem (similar to that of state assignment) and solved heuristically. The method was inexact because the associated output encoding problem could not be solved exactly.

In this paper, we present an exact algorithm for output encoding. The algorithm finds an encoding that minimizes the number of product terms in an optimized PLA implementation. The algorithm consists of the following steps:

1. Generation of generalized prime implicants (GPIs) from the original symbolic cover.

2. Solution of a constrained covering problem involving the selection of a minimum number of GPIs that form an encodeable cover.

3. Encoding of the symbolic outputs respecting the encoding constraints generated during Step 2.

4. Given the codes of the symbolic outputs and the selected GPIs, a PLA with product term cardinality equal to the number of GPIs can be trivially constructed. This PLA represents an exact solution to the encoding problem.

Various techniques to generate GPIs that are modifications to classical prime implicant generation techniques can be used in Step 1. The covering problem of Step 2 is more complex than the unate covering problem and hence classical covering algorithms cannot be directly used. Step 3 involves constrained encoding where the objective is to minimize the number of encoding bits required to satisfy the constraints. This step is also NP-complete. However, our focus here is to exactly minimize PLA product term cardinality and heuristically minimize PLA area.

We have also developed an exact state assignment algorithm that has essentially the same structure as the above procedure. In the state assignment case, the present states are represented as different values of a single multiple-valued variable (as in [14]). The covering problem is more complex than in the output encoding case and so is the constrained encoding problem. We use the formulation of [6] to pose the problem of four-level Boolean minimization as one of input-output encoding and give an exact solution similar to our state assignment algorithm.

In Section 2, basic definitions and notations used are given. The exact output encoding algorithm is described in Section 3. We give theorems that prove the correctness of the procedure. The procedure is generalized for the problems of state assignment and four-level Boolean minimization in Sections 4 and 5. Pruning heuristics that can be used in the exact solution of the different covering problems resulting in the output encoding, state assignment and four-level Boolean minimization cases are described in Section 6. Techniques for the creation of reduced prime implicant tables are also described. Heuristics to minimize the number of encoding bits used are touched upon in Section 7. Preliminary experimental results are presented in Section 8. In Section 9, we propose some computationally efficient heuristics based on the exact minimization algorithms. Finally, in Section 10, we describe how symbolic don't cares in output encoding can be handled.

2 Preliminaries

Let $B = \{0, 1\}, Y = \{0, 1, 2\}$. A **logic** (Boolean, switching) **function** ff in n input variables, $x_1, x_2, .. x_n$, and m output variables, $y_1, y_2, .. y_m$, is a function

$$ff : B^n \rightarrow Y^m$$

where $x = [x_1, .. x_n] \in B^n$ is the **input** and $y = [y_1, .. y_m] \in Y^m$ is the **output** of ff. B^n is the Boolean n-space associated with the function ff. Note that in addition to the usual values of 0 and 1, the outputs y_i may also take the **don't care value** 2 (or $-$) Such functions are called **incompletely specified** logic functions. A **completely specified function** f is a logic function taking values in $\{0, 1\}^m$, i.e., all the values of the input map into 0 or 1 for all the components of f. For each component of an incompletely specified logic function ff, ff_i, $i = 1, .. m$, one can define: the **ON-set**, $X_i^{ON} \subset B^n$, the set of input values x such that $ff_i(x) = 1$, the **OFF-set**, X_i^{OFF}, the set of values such that $ff_i(x) = 0$ and the **don't care** set X_i^{DC}, the set of values such that $ff_i(x) = 2$. A logic function with $m = 1$ is called a **single-output** function, while $m > 1$, it is called a **multiple-output** function.

A **cube** in a Boolean n-space associated with a logic function, f, can be specified by its vertices and by an index indicating to which components of f it belongs. An input cube c is specified by a row vector $c = [c_1, .. c_n]$ where each input variable takes on one of three values 0, 1 or 2 (or $-$) A 2 in the cube is a don't care input, which means that the input can take the values of either 0 or 1. For example, the cube 002 is equal to the union of the cubes 001 and 000. A **minterm** is a cube with only 0 and 1 entries. Cubes can also be classified based on the number of 2 entries in the cube. A cube with k entries or bits which take the value 2 is called a **k-cube**. A minterm thus is a **0-cube**.

A cube c_1 is said to **cover** (contain) another cube c_2, if each entry of c_1 is equal to the corresponding entry of c_2 or is equal to 2. The **supercube** of a set of cubes is the smallest cube that covers each cube in the set. A minterm m_1 is said to **dominate** another minterm m_2 (written as $m_1 \supset m_2$) if for each bit position in the second minterm that contains a 1, the corresponding bit position in the first minterm also contains a 1. Minterm m_2 is dominated by m_1 (written as $m_2 \subset m_1$). The **conjunction** of two minterms is the bitwise OR (written as $|$ or $\cup$) of the two minterms. The **disjunction** of two minterms is the bitwise AND (written as $\cap$) of the two minterms.

A logic function may have **multiple-valued** or symbolic input variables and symbolic output variables as in Figure 1. A symbolic input or output variable takes on **symbolic values**.

A finite state machine is represented by its **State Transition Graph** (STG) or **State Transition Table** (STT), $G(V, E, W(E))$ where V is the set of vertices corresponding to the set of states S, where $||S|| = N_S$ is the cardinality of the set of states of the FSM, an edge (v_i, v_j) joins v_i to v_j if there is a primary input that causes the FSM to evolve from state v_i to state v_j, and $W(E)$ is a set of labels attached to each edge, each label carrying the information of the value of the input that caused that transition and the values of the primary outputs corresponding to that transition. In general, the $W(E)$ labels are cubes or minterms.

3 Output Encoding

3.1 Introduction

The output encoding problem entails finding binary codes for symbolic outputs in a switching function as as to minimize the area or an estimate of the area of the encoded and optimized logic function. Here, we are concerned with two-level or PLA implementations of logic and hence the optimization step that follows encoding is one of two-level Boolean minimization.

An arbitrary output encoding of the function shown in Figure 1(b), is shown in Figure 2(a). The encoded cover is now a multiple-output logic function. This function can be minimized using standard two-level logic minimization algorithms. These algorithms exploit the sharing between the different outputs so as to produce a minimum cover. It is easy to see that an encoding such as the one in Figure 2(b), where each symbolic value corresponds to a separate output, can have no sharing between the outputs. Optimizing the function of Figure 2(b) would produce a function with a number of product terms equal to the total number of product terms produced by *disjointly* minimizing each of the ON-sets of the symbolic values of Figure 1(b). This cardinality is typically far from the minimum cardinality achievable via an encoding that maximally exploits sharing relationships.

3.2 Review of Previous Work

Some heuristic approaches to solving the output encoding problem have been taken in the past (e.g. [13], [17]). The program CAPPUCINO [13] attempts to minimize the number of product terms in a PLA implementation and secondarily the number of encoding bits.

The algorithm in CAPPUCINO is based on exploiting *dominance* relationships between the

```
0001  001              0001  10000
00-0  010              00-0  01000
0011  010              0011  01000
0100  011              0100  00100
1000  011              1000  00100
1011  100              1011  00010
1111  101              1111  00001

   (a)                    (b)
```

Figure 2: Possible Encodings of the Symbolic Output

binary codes assigned to different values of a symbolic output. For instance, in the example of Figure 1(b), if the symbolic value *out*1 is given a binary code 110 which dominates the binary code 100 assigned to *out*2, then the input cubes corresponding to *out*1 can be used as don't cares for minimizing the input cubes of *out*2. Using these don't cares can reduce the cardinality of the ON-set of the symbolic value *out*2. In CAPPUCINO, dominance relationships between symbolic values that result in maximal reduction of the ON-sets of the dominated symbolic values are heuristically constructed. Satisfying these dominance relationships (which should not conflict) results in some reduction of the overall cover cardinality. Minimum cardinality cannot be guaranteed because all possible dominance relations are not explored, nor is an optimum set selected. A more basic shortcoming is that dominance relations are not the only kind of relationships between symbolic values that can be exploited. After a symbolic cover has been encoded, it represents a multiple-output logic function and minimizing a multiple-output function entails exploiting other sharing relationships than just dominance.

3.3 Conjunctive Relationships

Consider the symbolic cover of Figure 3(a). The function has one symbolic output and one binary-valued output. Using dominance relationships alone in an encoding, it is not possible to reduce the size of any of the ON-sets of the symbolic values. One such encoding is shown in Figure 3(b), with *out*1 given the binary code 00, *out*2 given 01 and *out*3 given 11. However, if we code *out*1 with 11, *out*2 with 01 and *out*3 with 10 as in Figure 3(c), we obtain a reduction in cover cardinality after minimization (Figure 3(d)). Note that in a dominance relationship, the ON-set of the *dominated* symbolic value is reduced. However, in Figure 3(c) and 3(d), it is in fact the *dominating* symbolic value, *out*1, whose ON-set cardinality has been reduced from 1 to 0. This is because of the *con-*

```
101  out1 1
100  out2 1
111  out3 1
```

(a)

```
101  00 1
100  01 1
111  11 1
```

(b)

```
101  11 1
100  01 1
111  10 1
```

(c)

```
10-  01 1
1-1  10 1
```

(d)

Figure 3: Dominance and Conjunctive Relationships

junctive relationship between the codes of *out*2, *out*3 and *out*1. *out*1 $=$ *out*2 | *out*3 and hence the ON-set of *out*1 can be reduced using the ON-set of *out*2 and *out*3. Just making *out*1 dominate *out*2 and *out*3 is not enough, the code of *out*1 has to be the conjunction (bitwise OR) of the codes of *out*2 and *out*3. Exploiting these relationships is basic to a multiple-output logic minimizer and hence an exact encoding algorithm has to take into account these relationships in order to produce a minimum cardinality cover after optimization. Conjunctive relations may involve any number of symbolic values. For instance, the code of a symbolic value may be the bitwise OR of three other symbolic value codes.

Enumerating dominance or conjunctive relationships is very time-consuming. Finding the reduction in cover cardinality that can be accrued via an encoding satisfying each dominance or conjunctive relationship requires an exact logic minimization. Also, these relationships interact in complex ways and their effects are not simply cumulative. To solve the output encoding problem efficiently and exactly, we have to modify the prime implicant generation and covering strategies that are basic to two-level Boolean minimization.

3.4 An Exact Algorithm for Output Encoding

In this section, we present an exact algorithm for output encoding that guarantees a minimum cardinality encoded cover. As described briefly in Section 1, this algorithm is a four-step procedure. These steps are described in detail in the remainder of the section.

We are given a symbolic cover S with a single symbolic output (see Section 3.6 for generalization to the multiple symbolic output case). The different symbolic values are denoted v_1, .., v_N. The ON-sets of the v_i are denoted C_i. Each C_i is a set of D_i minterms $\{m_{i1}, .. m_{iD_i}\}$. Each minterm

1101	out1
1100	out2
1111	out3
0000	out4
0001	out4

(a)

1101	(out1)	110-	(out1, out2)
1100	(out2)	11-1	(out1, out3)
1111	(out3)	000-	(out4)
~~0000~~	~~(out4)~~		
~~0001~~	~~(out4)~~		

(b)

Figure 4: Generation of Generalized Prime Implicants

m_{ij} has a tag as to what symbolic value's ON-set it belongs to. Note that a minterm can only belong to a single symbolic value's ON-set. Minterms are commonly called 0-cubes.

3.4.1 Generation of Generalized Prime Implicants

The generation of generalized prime implicants (GPIs) proceeds as in the well-known Quine-McCluskey (Q-M) procedure [12], with some differences.

1-cubes are constructed by merging all pairs of mergeable 0-cubes. If two 0-cubes with the same tag, (v_i), are merged then the 1-cube has the same tag (v_i). On the other hand, if a 0-cube of tag (v_i) is merged with a 0-cube with tag (v_j), the resultant 1-cube has a tag $(v_i,\ v_j)$. The rule for canceling 0-cubes covered by 1-cubes is also different from the Q-M method. A 0-cube can be canceled by a 1-cube if and only if their tags are identical. A 1-cube 11− with tag $(v_1,\ v_2)$ cannot cancel a 0-cube 110 with tag (v_1).

The above can be generalized to the k-cube case.

1. When two k-cubes merge to form a $k+1$-cube, the tag of the $k+1$-cube is the *union* of the two k-cube tags.

2. A $k+1$-cube can cancel a k-cube only if the $k+1$-cube covers the k-cube and they have identical tags.

A cube with a tag that contains all the symbolic values $(v_1,\ ..,\ v_N)$ can be discarded and is not a GPI. These cubes are not required in a minimum solution (Theorem 3.3). The generation of generalized prime implicants for the symbolic cover of Figure 3(a) is shown in Figure 4. We have x GPIs with associated tags.

3.4.2 Selecting a Minimum Encodeable Cover

Given all the GPIs, we have to select a minimum subset of GPIs such that they cover all the minterms and form an *encodeable* cover. If we did not have the additional restriction of encodeability for a selected subset of GPIs, then the output encoding problem would be equivalent to two-level Boolean minimization. The selection is carried out by solving a covering problem (Section 6 deals with the covering problem). In the sequel, we describe what an encodeable cover means.

Consider a minterm, m, in the original symbolic cover S. Let the minterm belong to the ON-set of v_m. Obviously, in any encoded cover the minterm m has to assert the code given to v_m, namely $e(\ v_m\ )$. Let the selected subset of GPIs be p_1, .., p_G. Let the GPIs that cover m in this selected subset be $p_{m,1}$, .., $p_{m,M}$. For functionality to be maintained, the following relation has to be satisfied, for all minterms $m \in S$.

$$\bigcup_{i=1}^{M} \bigcap_{j} e(\ v_{p_{m,i},\ j}\ ) \quad = \quad e(\ v_m\ ) \quad \forall\, m \tag{1}$$

where the $v_{p_{m,i},\ j}$ represent the symbolic values that are in the tag of the GPI $p_{m,i}$. In Figure 5, we have a selection of GPIs for the symbolic cover of Figure 4(a) (whose GPIs are enumerated in Figure 4(b)). We have selected the GPIs 110−, 11 − 1 and 000− from Figure 4(b) in Figure 5(a). The constraints corresponding to Eqn. 1 for each minterm are given in Figure 5(b). The minterm 1101 is covered by both selected GPIs, one of which has a tag (*out*1, *out*2) and the other has a tag (*out*1, *out*3). Therefore, Eqn. 1 specifies

$$e(out1) \cap e(out2) \bigcup e(out1) \cap e(out3) \quad = \quad e(out1)$$

for the minterm xxx and other constraints for the remaining minterms. If a minterm is covered by a GPI with the same tag as the minterm, then the constraint specified by the minterm via Eqn. 1 is an identity.

Eqn. 1 gives a set of constraints on the codes of the symbolic values, given a selection of GPIs. If an encoding can be found that satisfies all these constraints, then the selection of GPIs is encodeable. However, a selection of GPIs may have an associated set of constraints that are mutually conflicting.

```
110-  (out1, out2)
11-1  (out1, out3)
000-  (out4)
```

(a)

1101 $e(out1) \sqcap e(out2) \bigsqcup e(out1) \sqcap e(out3) = e(out1)$
1100 $e(out1) \sqcap e(out2) = e(out2)$
1111 $e(out1) \sqcap e(out3) = e(out3)$
0000 $e(out4) = e(out4)$
0001 $e(out4) = e(out4)$

(b)

Figure 5: Encodeability of Selected GPIs

3.4.3 Dominance and Conjunctive Relationships to Satisfy Constraints

The constraints specified by Eqn. 1 can be satisfied by means of dominance and conjunctive relations between symbolic values. Continuing with our example, to satisfy

$$e(out1) \cap e(out2) \bigcup e(out1) \cap e(out3) = e(out1)$$

one has three alternatives:

1. $e(out2) \supset e(out1)$
2. $e(out3) \supset e(out1)$
3. $e(out1) \subseteq e(out2) \mid e(out3)$

Given an arbitrary constraint, a set of dominance and conjunctive relationships can be derived such that satisfying any single relation satisfies the constraint. Dominance and conjunctive relationships may conflict across a set of constraints. For instance, one cannot satisfy both $e(out1) \supset e(out2)$ and $e(out2) \supset e(out1)$. This represents a cycle in the dominance graph. Also, if one picks the equality in choice (3) above, then we require $e(out1) \supset e(out2)$ and $e(out1) \supset e(out3)$. In that case, one cannot satisfy both (1) and (3) with the same encoding.

Given a selection of GPIs, we derive a set of constraints via Eqn. 1 and construct a graph where each node represents a symbolic value. Directed edges in the graph represent dominance relations and undirected edges enclosed by arcs represent conjunctive relations. Each directed edge and arc has a label, corresponding to the minterm that produces the constraint represented by the edge or arc. The graph corresponding to the selected GPIs of Figure 5 is shown in Figure 6(a). A directed edge from *out*1 to *out*2 implies the code of *out*1 should dominate the code of *out*2. The dotted arc around the two undirected edges emanating from *out*1 implies that the code of *out*1 should be equal to or be dominated by the conjunction (bitwise OR) of the codes of its fanout symbolic values, in this case, *out*2 and *out*3. That is, $e(out1) \subseteq e(out2) \mid e(out3)$. *out*1 is called the **parent** in the conjunctive arc and *out*2 and *out*3, the **siblings** in the conjunctive arc. The conjunctive arc specifies equality or dominance, however, due to other relationships equality may be specifically required (see Lemma 3.4). In the case of conjunctive dominance the edges will be undirected, in the case of conjunctive equality the edges will directed towards the siblings to indicate that the parent dominates the siblings.

The graph corresponding to a selection of GPIs is encodeable and logic functionality is maintained, if two conditions are met. One selects either an edge or an arc of each label. In the case of selecting an arc, all dominance edges covered by the arc (implied by the conjunctive relationship) are also selected. For some selection,

1. There should be no directed cycles in the graph.

2. The siblings in any conjunctive equality arc should not have directed paths between each other.

3. No two conjunctive equality arcs can have exactly the same siblings and different parents.

4. The parent of a cojunctive equality arc should not dominate any symbolic value/node that dominates all the siblings in the arc.

The graph of Figure 6(b), derived from the graph of Figure 6(a), satisfies these properties and hence the selection of GPIs is valid. This implies that we can find an encoding such that the optimized cover has 2 product terms.

Given a constraint specified by Eqn. 1 of the form

$$a \cap b \cap c \ \bigcup \ a \cap d \cap e \ \bigcup \ a \cap f \cap g \ = \ a \qquad (2)$$

we have more complex choices than the equation in our example. To satisfy $a \cap b \cap c \ = \ a$, for instance, we need to satisfy both $b \supset a$ and $c \supset a$. This merely corresponds to a pair of directed edges that have to be selected simultaneously. Further, one can satisfy $a \cap b \cap c \ \bigcup \ a \cap d \cap f \ = \ a$ by satisfying $b \cap c \ \bigcup \ d \cap f \ \supseteq \ a$. This corresponds to a conjunctive relationship with nested

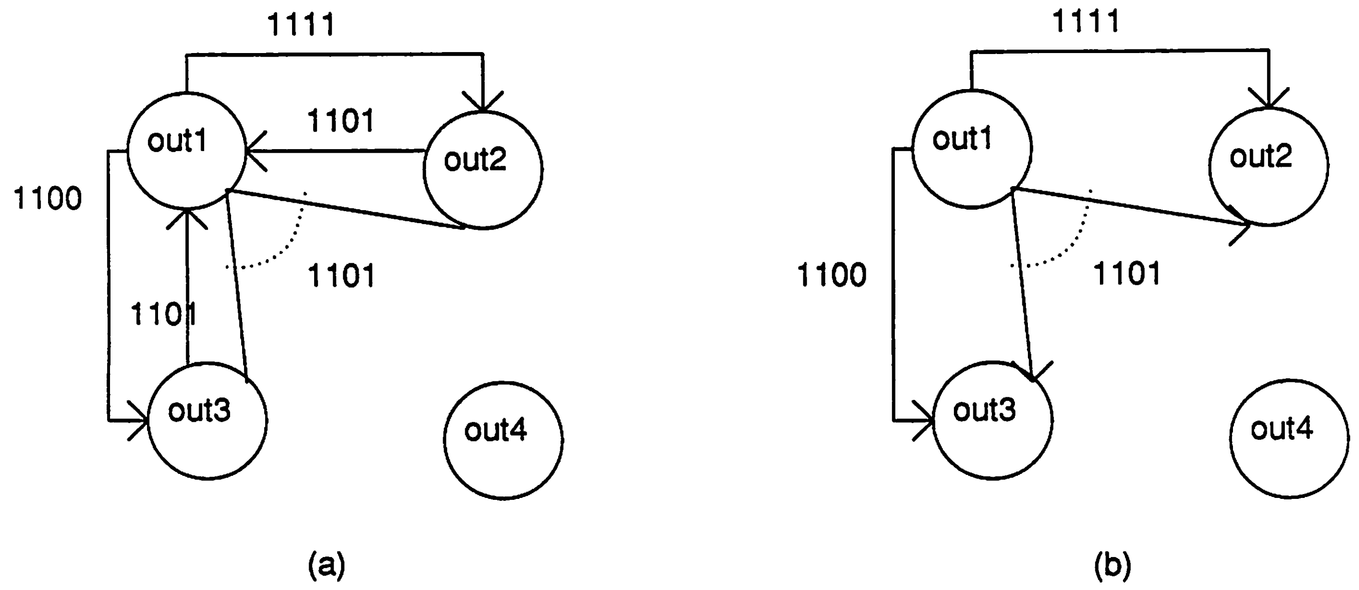

Figure 6: Encodeability Graphs

disjunctive terms. The siblings here are **disjunctive nodes** $b \cap c$ and $d \cap f$. These disjunctive nodes are dominated by b, c and d, f respectively. Conditions 2-4 should be satisfied for arcs whose siblings are disjunctive nodes as well. The symbolic values whose disjunction forms the disjunctive node are called the **ancestors** of the node. The ancestors dominate the disjunctive node. Also, if all the ancestors dominate a particular symbolic value, then the disjunctive node also dominates that value. For instance, if we have all the ancestors of a disjunctive node dominating the parent of a conjunctive arc that the node is a sibling of, then we have a cycle in the graph rendering it unencodeable.

3.4.4 Constructing the Optimized Cover

If a selection of GPIs has been made that covers all minterms and is encodeable, then an encoding can be trivially found that satisfies the constraints (see Theorem 3.5). We can now construct an encoded and optimized cover. The cover will contain the selected GPIs. For each GPI, the output combination in the cover is found using the tag corresponding to the GPI. The codes corresponding to all the symbolic values in the tag of the GPI are intersected (bitwise ANDed) to produce the output part. Continuing with our example, the GPIs selected and the tags for the GPIs are shown in Figure 7(a). These GPIs have an associated graph that is encodeable and an encoding satisfying the constraints is given in Figure 7(b). Note that the encoding has to satisfy conjunctive equivalence $e(out1) = e(out2) \mid e(out3)$, rather than conjunctive dominance $e(out1) \subset e(out2) \mid e(out3)$.

110-	(out1, out2)
11-1	(out1, out3)
000-	(out4)

(a)

out1	-> 11
out2	-> 01
out3	-> 10
out4	-> 00

(b)

110-	01
11-1	10
000-	00

(c)

Figure 7: Consructing the Optimized Cover

This is because of the dominance relationships $e(out1) \supset e(out2)$ and $e(out1) \supset e(out3)$. We have constructed the optimized cover with the GPIs by intersecting the codes of symbolic values in the tags of each GPI to obtain the output part (Figure 7(c)).

3.5 Correctness of Procedure

Proposition 3.1 *The selection of a minimum cardinality encodeable cover from the GPIs represents an exact solution to the output encoding problem.*

In the remainder of this section, we will justify Proposition 3.1. First, we show that logic functionality is retained.

Lemma 3.2 *Satisfying Eqn. 1 and constructing the output part as in Section 3.4.4 retains logic functionality.*

Proof: We construct the output part of a GPI by intersecting all the codes of the symbolic values contained in its tag. That is precisely the intersection term in Eqn. 1. The output of a minterm in a PLA is the OR of all the outputs asserted by the cubes that cover the minterm. This corresponds to the union (OR) in Eqn. 1. Thus, satisfying Eqn. 1 implies that each minterm asserts the same output combination as it would have in the original encoded but unoptimized cover. **Q.E.D.**

Next, we show that the canceled k-cubes during GPI generation are not necessary in a minimum solution.

Theorem 3.3 *A minimum cardinality encodeable solution can be made up entirely of GPIs.*

Proof: Assume that we have a minimum cardinality solution with a cube c_1 that is not a GPI. Let the tag of c_1 be T. We know a GPI p_1 exists such that $p_1 \supset c_1$ and such that the tag of p_1 is T. Replacing c_1 with p_1 will not change the cardinality of the cover. The minterms corresponding to $p_1 - c_1$ will be covered by an extra GPI p_1 and therefore Eqn. 1 for those minterms will be different. However, the extra tag in the equation merely represents an extra option in the graph corresponding to the encodeability. Since the original graph was encodeable, adding edges with the same label as the labels of edges originally contained in the graph will not change the encodeability.

We have also discarded cubes with tags that contain all the symbolic values. If such a cube exists in a minimum encoded cover, it asserts the output combination given by the intersection of the codes of *all* the symbolic values. If this intersection is null (all 0s), then the cube can be discarded to obtain a smaller cover. If the intersection is not null and the cube asserts some outputs, then it means that for the bits corresponding to these outputs, all the codes of the symbolic values have a 1. We can reduce the codes of the all the values and still maintain their identities by discarding these outputs. Then, the cube asserts a null output combination and can be discarded. Thus, the cube is not required in a minimum cover.

Hence, we have a minimum cardinality encodeable selection can be made up entirely of GPIs. **Q.E.D.**

Thus, if one selects a minimum set of GPIs that cover all minterms and have an associated set of constraints by Eqn. 1 that is encodeable, we are guaranteed a minimum solution to the encoding problem. It remains to prove that the conditions to the satisfied by the graph for encodeability are necessary and sufficient conditions.

Lemma 3.4 *If a conjunctive arc is required in a graph to produce a selection of choices that is encodeable, then the arc represents a conjuntive equality constraint.*

Proof: Without loss of generality, assume we have a constraint specified of the form of Eqn. 2. We require the choice a conjunctive arc of the form $b \cap c \bigcup d \cap f \supseteq a$, only if ($b \not\supset a$ or $c \not\supset a$) and ($d \not\supset a$ or $e \not\supset a$). Else, we can make $b \supset a$ and $c \supset a$ to satisfy $a \cap b \cap c = a$ or $d \supset a$ and $e \supset a$ to satisfy $a \cap d \cap e = a$. This implies we have to select an edge $a \supset b$ or $a \supset c$ and an edge $a \supset d$ or $a \supset e$. Say, we have to select $a \supset b$ and $a \supset d$. This means $b \cap c \subset a$ and $d \cap f \subset a$. Therefore, $b \cap c \bigcup d \cap f \not\supset a$ and we can only satisfy $b \cap c \bigcup d \cap f = a$. The same argument holds if we originally select edges $a \supset c$ and $a \supset e$ or any other combination. **Q.E.D.**

Theorem 3.5 *Conditions 1-4 stated in Section 3.4.3 are necessary and sufficient for the graph to be encodeable.*

Proof: *Necessity*: If the graph is cyclic, then we have two dominance relations $v_1 \supset v_2$ and $v_2 \supset v_1$ that cannot be satisfied simultaneously. If two siblings v_1 and v_2 of a conjunctive arc have a directed path between each other, it implies that $v_1 \supset v_2$ or $v_2 \supset v_1$. However, we require $v_1 \mid v_2 = v_3$. If $v_1 \supset v_2$ ($v_2 \supset v_1$) then $v_1 = v_3$ ($v_2 = v_3$), which is not allowed. If two conjunctive equality arcs with exactly the same siblings exist, the two parents will have to have the same code to satisfy both equalities. This is not allowed. If a graph violates Condition 4, then the conjunction of the siblings in the arc is dominated by some symbolic value that is dominated by the parent. This means that the parent both has to dominate and equal the conjunction of the siblings.

Sufficiency: Given a graph satisfying conditions 1-4, the procedure below constructs a correct encoding. The procedure assumes all conjunctive arcs are binary, but can be easily generalized.

We initially set the codes of all values to null. For each conjunctive arc, we code the siblings with 01 and 10 and the parent with 11, except or the cases 7 and 8, where more bits may be used. For all other values (or nodes):

1. If the node dominates a single sibling, we append that sibling's code to the node.

2. If the node dominates both siblings, we append the parent's code to the node.

3. If the node dominates parent, we append the parent's code to the node.

4. If the parent dominates the node, we append the parent's code to the node.

5. If a single sibling dominates the node, we append the code of the sibling to the node.

6. If both siblings dominate the node, we append the code of all zeros to the node.

7. If a node dominates a single sibling and is dominated by the parent, then we perform the following steps. We append the node and the sibling it doesn't dominate with 101, the dominated sibling with 100 and the parent with 111. In the case of K nodes forming a chain from parent to a sibling, we use $K + 2$ bits, code the parent with $K + 2$ 1s, topmost node (level 1) and non-dominated sibling with $K + 1$ 1s, a node at level i with $K + 2 - i$ 1s and the dominated sibling with one 1.
   A node cannot dominate both siblings and be dominated by the parent due to Condition 4.

8. If a sibling is a disjunctive node, then we have to code the ancestors correctly. If one of the ancestors is dominated by the parent, we are back to Case 7. If both ancestors are dominated by the parent, then we code the parent with 1111, one ancestor with 1010, the other with 0110 and the sibling gets the intersection 0010. The other sibling's ancestors are coded such that it gets the code 1101. If neither ancestor is dominated by the parent, then we code one

of them with 101, the other with 100, the sibling gets the intersection 100 and the ancestors of the other sibling are given codes such that the other sibling gets 011.

Next, for each conjunctive arc, the siblings, parent, all nodes dominated by the parent and dominating a sibling and all ancestors of disjunctive node siblings are merged into a composite node. The graph is levelized. If L is the number of levels in the graph, the topmost set of nodes is appended a code of length L with L 1s, the next level codes with $L-1$ 1s and so on. Appending a code to a composite node implies that all the nodes contained in the composite node are appended with the same code. **Q.E.D.**

3.6 Multiple Symbolic Outputs

The procedure outlined can be generalized to the case where we have multiple symbolic outputs. Each minterm initially has a number of tags equal to the number of symbolic outputs. Each tag corresponds to the symbolic value whose ON-set the minterm belongs to, for each symbolic output. Minterm pairs are merged and the operations on the tags are performed exactly the same as before. A $k+1$-cube cancels a k-cube only if all of its tags are identical to the corresponding tags of the k-cube. Cubes with tags such that all corresponding symbolic values are contained in the tag can be discarded. Thus, the GPIs can be generated. We have separate graphs representing encoding constraints for each symbolic output. Given a selection of GPIs, these graphs can be constructed and checked for encodeability as before. All the graphs have to be encodeable for a selection of GPIs to be valid.

The generalization to functions with both symbolic and binary-valued outputs is described in Section 4.2.

3.7 The Issue of the All Zeros Code

If a code of all zeros is given to a symbolic value, then it is possible that one or more GPIs can be dropped in a PLA implementation, from an otherwise minimum cover. This is because in a PLA implementation, one is only concerned with the ON-sets. The procedure presented has not taken this fact into account.

A solution is to perform $N+1$ minimizations where N is the number of symbolic values. One minimization is as before. In the other N minimizations, we drop all the minterms in the ON-set of each of the N symbolic values, one value's ON-set at a time. We select the best solution out of

0	S1	S1	1
1	S1	S2	0
1	S2	S2	0
0	S2	S3	0
1	S3	S3	1
0	S3	S3	1

Figure 8: State Transition Table of Finite State Machine

the $N+1$ minimizations. The reason we have to perform the first minimization without dropping any of the minterms is that the all zeros code cannot appear in conjunctive relations, since it is dominated by all other codes. Hence, constraining oneself to use a code of all zeros may result in a sub-optimal solution.

We can prove the following theorem which gives conditions where multiple minimizations are not required.

Theorem 3.6 *Given a cover with one or more symbolic outputs and binary-valued outputs if all minterms in the cover belong to the ON-set of at least one binary-valued output, then there can be no advantage to using an all zeros code.*

Proof: The only advantage in using an all zeros code is that minterms may be dropped by putting them into OFF-sets. We can always satisfy required dominance and/or conjunctive relationships via codes other than the all zeros code. In the case of a cover with the property mentioned above, we cannot drop any of the minterms. Hence, we can obtain a minimum cardinality solution without using the all zeros code. **Q.E.D.**

4 State Assignment

4.1 Introduction

The state assignment problem is an input-output encoding problem with equality constraints on the symbolic inputs and outputs. In Figure 8, a State Transition Table (STT) of a finite state machine (FSM) is shown. The present states (2nd column) can be viewed as a symbolic input and the next states (3rd column) can be viewed as a symbolic output.

An input encoding problem in isolation can be solved by representing the symbolic input as a multiple-valued variable [14], where each distinct symbolic value represents a distinct value of the multiple-valued variable. Exact minimization of the resulting multiple-valued function produces a

```
10 10000  1010
01 10000  0110              01 10000  0110
10 01000  1010              10 11000  1010
-1 01000  1011              -1 01000  1011
1- 00100  0110              1- 00100  0110
0- 00100  1001              0- 00100  1001
-- 00010  0010              -- 00010  0010
-- 00001  1101              -- 00001  1101

     (a)                         (b)
```

Figure 9: Multiple-Valued Functions

minimum cardinality multiple-valued cover. The symbolic cover of Figure 1(a) has been represented as a multiple-valued function in Figure 9(a). The symbolic value *inp*1 is the value 10000, *inp*2 is 01000 and so on. Minimizing the function produces the result of Figure 9(b). The merged input implicants in the minimized multiple-valued cover represent constraints that the binary codes assigned to the symbolic values have to satisfy, in order to produce an encoded binary cover with the same cardinality as the minimized multiple-valued cover. Any set of these *input constraints* can always be satisfied by some encoding.

A symbolic value is **contained** in a multiple-valued implicant if the position corresponding to the symbolic value has a 1 in the implicant. For instance, the symbolic values *inp*1 and *inp*2 are contained in the implicant 11000. The constraint specified is that the *supercube* of the codes of all the symbolic values contained in the implicant should not intersect any of the codes given to the symbolic values not in the implicant. In our example, it means that the smallest cube covering the codes assigned to *inp*1 and *inp*2 should not intersect the codes of *inp*3, *inp*4 and *inp*5. Each distinct multiple-valued implicant specifies a distinct constraint on the codes that can be assigned to the symbolic values. As mentioned previously, an encoding satisfying the constraints specified by any set of multiple-valued implicants can always be constructed.

To solve the state assignment problem exactly, one can treat the present state space as a multiple-valued variable and solve the resulting output encoding problem exactly. Modifications that are required to the strategy presented in Section 3 will be described in the remainder of this section.

4.2 Generation of Generalized Prime Implicants

We now have a function with multiple binary-valued inputs, a single multiple-valued input, one symbolic output and multiple binary-valued outputs, that is to be encoded. Each minterm has a tag corresponding to the symbolic next state whose ON-set it belongs to. Each minterm also has a tag that corresponds to all the outputs asserted by the minterm.

Two minterms or 0-cubes can merge to form a 1-cube. Merging may occur between minterms with the same binary-valued part and different multiple-valued parts or uni-distant binary-valued parts and the same multiple-valued part. The next state tag of the 1-cube is the union of the next state tags of the two minterms. As in the Q-M method, the binary-valued output tag of the 1-cube will contain only the outputs that both minterms asserted. A 1-cube can cancel a 0-cube if and only if their next state and binary-valued output tags are identical *and* their multiple-valued parts are identical. Thus, a 1-cube 1 011 (where the second term is a multiple-valued implicant) cannot cancel 1 001 even if their next state and output tags are identical. This is because the merging of the multiple-valued part represents an input constraint as described in Section 4.1. One exception is when the multiple-valued input part of the 1-cube contains all the symbolic states – in this case the implicant represents an input constraint that is satisfied by *any* encoding.

Generalizing to k-cubes, we have:

1. A $k+1$-cube formed from two k-cubes has a next state tag that is the union of the two k-cubes' next state tags and an output tag that is the intersection of the outputs in the k-cubes' output tags.

2. A $k+1$-cube can cancel a k-cube only if their multiple-valued input parts are identical or if the multiple-valued input part of the $k+1$-cube contains all the symbolic states. In addition, the next state and output tags have to be identical.

A cube with a next state tag containing all the symbolic states and with a null output tag can be discarded. The generation of GPIs for the FSM of Figure 8 is depicted in Figure 10. 13 GPIs are eventually produced.

4.3 Selecting a Minimum Encodeable Cover

Given all the GPIs, we select a minimum encodeable set that covers all minterms by solving a covering problem (Section 6), as before. However, the definition of encodeability is different due to the complication of having the input constraints.

0 100 (s1) (o1)	- 100 (s1, s2) ()	0 111 (s1, s3) ()
1 100 (s2) ()	~~0 110 (s1, s3) ()~~	- 011 (s2, s3) ()
1 010 (s2) ()	0 101 (s1, s3) ()	1 111 (s2, s3) ()
0 010 (s3) ()	1 110 (s2) ()	~~- 110 (s1, s2, s3) ()~~
~~1 001 (s3) (o1)~~	~~1 101 (s2, s3) ()~~	⋮
~~0 001 (s3) (o1)~~	- 010 (s2, s3) ()	
	~~1 011 (s2, s3) ()~~	
	0 011 (s3) ()	
	- 001 (s3) (o1)	

Figure 10: Generation of GPIs in State Assignment

An input constraint may conflict with dominance or conjunctive relations. Therefore, when we pick a set of GPIs, we need to check that the input constraints, given by the merging of the multiple-valued implicants, as well as the relations given by Eqn. 1 are compatible. In [13], the question of compatibility between input constraints and dominance relationships was posed and a theorem stating necessary and sufficient conditions for compatibility was given. Here, we have a more complex case of possibly mutually conflicting input, dominance and conjunctive relations. We can prove the following theorem.

Theorem 4.1 *Given a set of dominance and conjunctive relations represented by a graph and a set of input relations, a necessary and sufficient set of conditions for the existence of an encoding satisfying all the relations are:*

1. *Conditions 1-4 of Theorem 3.5 are satisfied.*
2. *For any state tuple $s1$, $s2$ and $s3$ such that $s1 \supset s2$ and $s2 \supset s3$, no input relation should exist such that the position corresponding to $s1$ and $s3$ has a 1 and the position corresponding to $s2$ has a 0.*
3. *No input relation should exist where all the siblings of a conjunctive equality arc have a 1 and the parent 0. In the case of the siblings being disjunctive nodes, no input relation should exist where one or more ancestors of each disjunctive sibling have a 1 and the parent 0.*

Proof: *Necessity*: The proof of the necessity of Condition 1 is identical to the proof in Theorem 3.5. Assume a set of relations violates Condition 2 and an encoding exists satisfying all relations. The encoding will have a bit in the codes of $s1$, $s2$ and $s3$ such that $s1/s3$ have 1s in the bit and $s2$ has a 0 or $s1/s3$ have 0s and $s2$ has a 1, to satisfy the input relation. In the former case, $s2 \not\supset s3$

and in the latter $s1 \not\supset s2$ and the output relations could not have been satisfied. The same is true in the disjunctive node case as well.

Assume an input constraint exists where all the siblings of a conjunctive equality arc have a 1 and the parent a 0. The supercube of the codes of the siblings of an equality arc always intersects the code of the parent. This means the input constraint cannot be satisfied if the output relations are satisfied.

Sufficiency: We will show how an encoding can be constructed if Conditions 1-3 are satisfied. We first construct an encoding satisfying the dominance and conjunctive relations the procedure of Theorem 3.5. Pick an arbitrary input constraint from the input constraint set. We group the states with 1s in the constraint into $Q1$ and the states with 0s into $Q0$. We are guaranteed via Condition 3 that if $q_1 \supset q_2$ and $q_2 \supset q_3$, then both q_1 and q_3 don't belong to $Q1$ or don't belong to $Q0$. If $q_1 \in Q1$ and $q_3 \in Q0$, then we append a column to the states' codes such that all $q \in Q1$ have a 1 and all $q \in Q0$ have a 0. If $q_1 \in Q0$ and $q_3 \in Q1$, then we append a column to the states' codes such that all $q \in Q0$ have a 1 and all $q \in Q1$ have a 0. This does not violate the dominance relations.

Next, we consider the conjunctive arcs. We have six possible cases, enumerated below.

1. Parent in $Q0$, all siblings in $Q1$: Cannot occur due to Condition 4.

2. Parent in $Q1$, all siblings in $Q0$: Without loss of generality, assume a binary conjunctive arc with two siblings. The initial encoding satisfies the equality constraint for the arc. We append to the code of all $q \in Q1$, including the parent the code 110. Hence, dominance relations within $Q1$ are not changed. The two siblings in $Q0$ are appended with 010 and 100. The siblings are never required to dominate each other. If any states in $Q0$ are required to dominate all or any of the siblings in $Q0$ or states in $Q1$, we can append the code 111 to those states. The states in $Q0$ that have to be dominated by any of the siblings or any of the states in $Q1$ can be appended with the code of the dominating sibling or in the case of multiple dominating siblings, the code 000.

3. Parent + some siblings in $Q1$, other siblings in $Q0$: We code the parent and siblings in $Q1$ with a 1 and the siblings in $Q0$ with a 0. If no other dominance relations between states in $Q0$ and $Q1$ exists, code the rest of the states in $Q0$ with a 0 and $Q1$ with a 1. Else, do the same as in Case 2, using more bits.

4. Parent + some siblings in $Q0$, other siblings in $Q1$: Consider a binary conjunctive arc. The

sibling in $Q0$ is appended with the code 01, the parent 11 and the sibling in $Q1$ with 10. We can maintain other dominance relations as before.

5. Parents + all siblings in $Q1$: Code all parents and siblings as well as the other states in $Q1$ uniformly with a 1 or a 0, depending on other dominance relations.

6. Parents + all siblings in $Q0$: Similar to Case 5.

Q.E.D.

4.4 Constructing an Optimized Cover

Once the GPIs have been selected and an encoding satisfying all relations is found, it a simple matter to construct the optimized cover. The output tag of each GPI gives the outputs asserted by the GPI. Intersecting the binary codes of all states in the next state tag gives the next state part (in binary form). The multiple-valued input part of a GPI is replaced by the supercube of the codes of all states in the multiple-valued implicant.

Arguments very similar to those of Section 3.5 can be used to show that the procedure described in this section does indeed result in a minimum solution to the state assignment problem.

5 Four-Level Boolean Minimization

5.1 Introduction

The problem of multi-level Boolean minimization aims at finding an optimum representation of a logic function as a cascade of two-level logic functions. The objective is to minimize the area of the eventual implementation. The problem we address is the following:
Given a two-level function, find an optimum decomposition of the function into two two-level functions such that the inputs to the first function are the original primary inputs, the inputs to the second function are outputs of the first function and the outputs of the second function are the original primary outputs. An optimum representation is defined a representation where a function of the number of product terms in the two-level functions is minimized.

Example decompositions are shown in Figure 11. Several points are worthy of note.

1. We consider that all the primary inputs (PIs) or a selected subset of PIs feed into the first function and all the primary outputs are asserted by the second function. However, in an

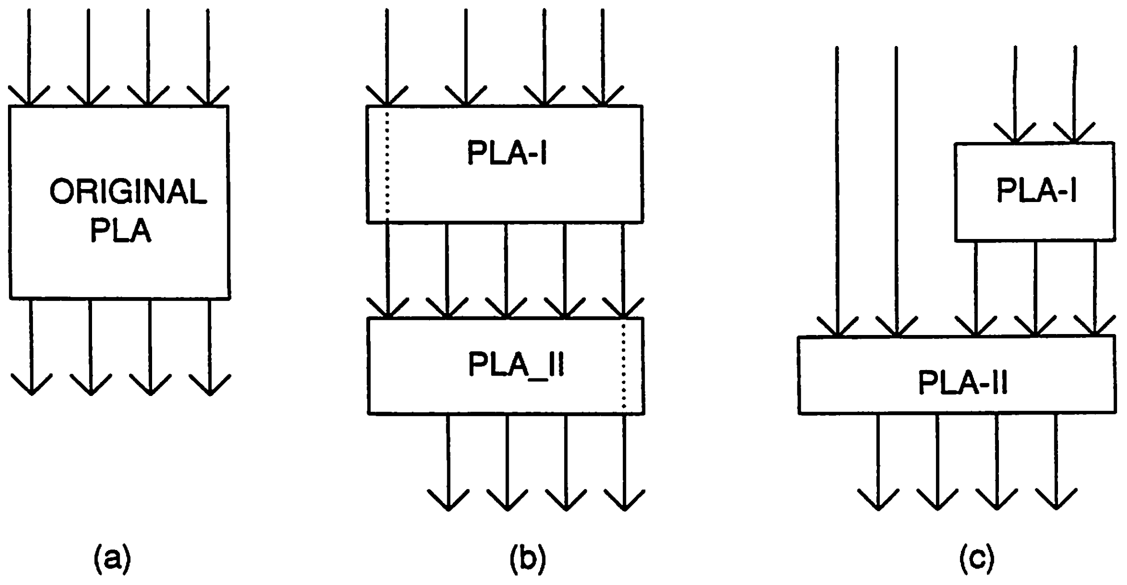

Figure 11: Logic Decomposition

optimum decomposition, an output of the first function may, in fact, be a direct connection to a primary input and be used in the second function, as shown in Figure 11(b). Similarly, a primary output may, in fact, be asserted by the first function and pass unchanged through the second function.

2. A subset of primary inputs may be initially specified for a decomposition as shown in Figure 11(c). This is the more general case of decomposition. Note that, even in this case, some intermediate lines (ILs) may, in reality, be PIs or POs.

3. The cost function that is optimized should ideally be the area of the decomposed functions, i.e. a function of the number of PIs, POs, ILs and product terms in each function or PLA. Since, we cannot easily estimate the number of ILs beforehand (it involves an encoding step), the cost function used here is a linear weighted sum of the number of product terms in the two PLAs (the cost function may be non-linear if required).

This problem was first addressed in [18]. In [6], a decomposition heuristic based on multiple-valued minimization was proposed. Given a two-level cover and a subset of selected inputs, the algorithm in [6] performs the following steps:

1. The two-level cover is made disjoint in the selected subset of inputs. This identifies a set of disjoint input combinations for the selected subset. The combinations may be cubes or minterms.

2. A PLA with the input combinations represented as values of a symbolic input and asserting the original outputs is constructed. This PLA is called the **driven** PLA.

3. A **driving** PLA with the original input combinations producing a symbolic output is constructed. The symbolic values asserted by the symbolic output of the driving PLA and the symbolic values taken by the symbolic input of the driven PLA have a one-to-one correspondence.

4. The driving and driven PLA form a cascade. We have now an input-output encoding problem. The problem is approximated as an input encoding problem for the driven PLA and solved using multiple-valued minimization.

This algorithm is heuristic because the size of the driving PLA is not taken into account – the output encoding problem for the driving PLA is completely ignored.

In the next section, we describe how the state assignment algorithm described in Section 5 can be modified to the four-level Boolean minimization case.

5.2 Modifications

There are two important differences between the four-level Boolean minimization problem addressed here and the state assignment problem. Firstly, in the Boolean minimization problem, we have two distinct covers rather than a single one. Our goal is to minimize a linear weighted sum of the two cover cardinalities. The second difference is more subtle. The combinations corresponding to the selected inputs become values of a symbolic input to the driven PLA which are to be re-encoded. If one symbolic value always asserts the same primary output as another values (for the different unselected input combinations), these two values can have the same code in the driving or driven PLAs. Constraining them to have the same code or constraining them to be different may result in a sub-optimal solution to the output encoding problem and therefore for the Boolean minimization problem. We have to use this extra degree of freedom in an optimum way.

The generation of generalized prime implicants (GPIs) is separate for the two covers. For the driving PLA with the symbolic output, GPIs are generated as described in Section 3.4.1. The symbolic input in the driven PLA is replaced by a multiple-valued variable. The generation of GPIs is similar to the state assignment case – a $k+1$-cube cancels a k-cube only if the multiple-valued input parts and the output tags are identical (the binary-valued input parts will differ).

We thus have two sets of GPIs and we have to select a subset from each of the two sets such that the subsets cover all the minterms in each cover and *together* form an encodeable cover. Like in the state assignment case, we have equality constraints between the symbolic values representing the same selected input combination. The compatibility between the input relations given by the selected subset of GPIs for the driven PLA and the output relations given by the selected subset of GPIs for the driving PLA is determined via the conditions of Theorem 4.1. A difference is that some symbolic values may be allowed to have the same code and hence Conditions 1-4 of Theorem 3.5 may be relaxed for these values. For example, cycles are permitted within these values alone and these values can be parents of a conjunctive arcs with exactly the same sets of siblings.

The covering problem to be solved in the output encoding, state assignment and four-level Boolean minimization cases is described in the next section.

6 Solving the Covering Problem

6.1 Introduction

The classical covering problem of two-level Boolean minimization involves finding a minimum set of prime implicants (PIs) that form a cover for a logic function. Here, we have the additional restriction on the selected generalized prime implicants (GPIs) – they have to form an *encodeable* cover. The definition of encodeability varies for the output encoding, state assignment and four-level Boolean minimization cases. However, the covering algorithm need only be concerned with a black box that determines encodeability of the selected set of GPIs and a few other properties of the constraint graph associated with the selected GPIs (Section 6.4).

In Section 6.2, we first describe how various techniques for generating the prime implicants of binary-valued output functions can be used to generate all the GPIs for functions with symbolic outputs. In Section 6.3, we review strategies for solving the classical covering problem and in Section 6.4 we describe our approach to solving the covering problem with associated encodeability constraints.

6.2 Reduced Prime Implicant Table Generation

Many techniques for determining all the PIs of single and multiple-output logic functions have been published in the past [12] [19]. An algorithm based on the recursive decomposition of a function followed by a pairwise consensus operation has been reported [3] and has been improved upon in

0001	out1
00-0	out2
0011	out2
0100	out3
1000	out3
1011	out4
1111	out5

(a)

0001	11110
00-0	11101
0011	11101
0100	11011
1000	11011
1011	10111
1111	01111

(b)

Figure 12: Transformation for Output Encoding

the program McBOOLE [4]. Other techniques have been reported in [15]. These techniques not only efficiently generate PIs without duplication of effort but also create a *reduced prime implicant table.* In the prime implicant table of the Q-M algorithm, each column in the table corresponds to a minterm of the function and each row to a PI. In a reduced prime implicant table, each column corresponds to a collection of minterms (i.e. a larger subspace), all of which are covered by the same set of PIs. Thus, using the algorithms of [15] for example, rather than the Q-M method leads to a more efficient creation of the prime implicant table.

We cannot directly use these techniques on functions with symbolic outputs to generate all GPIs. The canceling rule for GPIs is not the same as the canceling rule for PIs. However, we can transform a function with a symbolic output into a function with multiple binary-valued outputs such that the PIs for this new multiple-output function have a one-to-one correspondence with the GPIs of the original function. This is illustrated in Figure 12. The function with a symbolic output of Figure 1(b) has been duplicated in Figure 12(a). Each symbolic value is replaced by an output combination to produce the binary-valued multiple-output function of Figure 12(b). M outputs are required if there are M symbolic values. A symbolic value has an output combination of all 1s and one 0 in a unique identifying position. These outputs perform the same function as the output tag in GPI generation (Section 4.1).

Lemma 6.1 *The PIs of the function obtained via the transformation described are the GPIs of the original function with the symbolic output.*

Proof: The set of outputs asserted by any cube in the new function is the set of symbolic values *not* in the tag of the corresponding cube in the original function. While generating the PIs for the binary-valued multiple-output function, a cube, c_1, cancels another cube, c_2, only if c_1 covers

0	S1	S1	1
1	S1	S2	0
1	S2	S2	0
0	S2	S3	0
1	S3	S3	1
0	S3	S3	1

(a)

0 001	1 110 110
1 001	0 101 110
1 010	0 101 101
0 010	0 011 101
1 100	1 011 011
0 100	1 011 011

(b)

Figure 13: Transformation for State Assignment

c_2 and the outputs asserted by c_1 are the same as the outputs asserted by c_2. This implies that the set of symbolic values in the tag of the two corresponding cubes in the original function are identical and c_1 would have canceled c_2 there as well. Finally, cubes in the binary-valued function formed with a null output combination are discarded. This corresponds to discarding cubes with tags containing all the symbolic values. **Q.E.D.**

Thus, via this transformation we can make use of the classical techniques for prime implicant generation. In the state assignment case, we have a symbolic or multiple-valued input variable. We also have the restriction during GPI generation that the multiple-valued part of a $k+1$-cube that cancels a k-cube has to be identical. This does not apply to PI generation in multiple-valued input, binary-valued output functions [16]. We thus have a more complex transformation in the case of a function with symbolic input and output. This transformation is illustrated in Figure 13. In Figure 13(a), we have duplicated the State Transition Table (STT) of Figure 8. The new function of Figure 13(b) has three sets of binary-valued outputs. The first set corresponds to the original binary-valued outputs in the STT. The second set corresponds to the next states. Given N_S states, we have N_S binary-valued outputs in this set. This set performs the function of the next state tag in GPI generation (Section 4.2). The third and last set of outputs incorporates the restriction of the equality of the multiple-valued input parts for cube cancellation. This set of N_S outputs corresponds to the present state space. It is constructed like the second set – each state has a unique N_S-bit code with $N_S - 1$ 1s and one 0.

The argument that generating the PIs for this transformed function is equivalent to generating the GPIs for the original function follows in a similar way to the proof of Lemma 6.1.

In the four-level Boolean minimization case, we generate the GPIs for the driving PLA by transforming it as in Figure 12. The driven PLA has a symbolic input and binary-valued outputs.

We append a set of outputs corresponding to the symbolic input like the present state set (in the state assignment case) and generate the PIs for the transformed function.

Once the GPIs have been generated, the additional outputs are discarded, since we have to solve a different covering problem from the standard covering problem. The next state tags and/or output tags for each GPI are constructed by finding all the symbolic values whose ON-sets intersect the GPI.

6.3 The Classical Covering Problem

The standard branch-and-bound solution to the minimum cover problem involves the following steps (rows correspond to PIs and columns to collections of minterms):

1. Remove columns that contain other columns and remove rows which are contained by other rows. Detect essential rows (a column with a single 1 identifies an essential row) and add these to the selected set. Repeat until no new essential elements are detected.

2. If the size of the selected set exceeds the best solution thus far, return from this level of recursion. If there are no elements left to be covered, declare the selected set as the best solution recorded thus far.

3. Heuristically select a branching row.

4. Add this row to the selected set and recur for the sub-table resulting from deleting the row and all columns that are covered by this row. Then, recur for the sub-table resulting from deleting this row without adding if to the selected set.

In [15], a lower bounding technique based on a maximal independent set heuristic was proposed. In Step 2, a maximal set of columns, all of which are pairwise disjoint is found using a straightforward, greedy algorithm (Finding a *maximum* independent set of columns is itself NP-complete). Because each column must be covered and all the columns in the maximal independent set share no row in common, the size of the maximal independent set is a lower bound on the number of rows required to complete the cover. At Step 2, the recursion can be bounded if the size of the selected set at Step 2 *plus* the size of the maximal independent set equals or exceeds the best solution known.

6.4 Covering with Encodeability Constraints

The algorithm we use is a modification of the algorithm described in the previous section. The modifications are described in the sequel.

In Step 1, a row (GPI) is deemed to contain another row (GPI) only if the tags of the two GPIs are identical or the tag of the first GPI is a subset of the tag of the second (This may happen lower in the recursion after some columns have been deleted). The lower bounding criterion at Step 3 uses the size of the maximal independent set of columns. This bound is looser than in standard covering because even if a cover can be constructed with a number of elements equal to the lower bound, it may not be encodeable.

Once the selected set covers all elements, we perform an encodeability check. If the cover is encodeable, we declare the solution as the best recorded until then. If not, we perform another branch-and-bound step to find the *minimum* number of GPIs (rows) which when added to the selected set renders it encodeable. The GPIs during this branch-and-bound step are selected from the current sub-table in the recursion. This branch-and-bound step is now described.

1. If the selected set is encodeable, then declare the selected set as the best encodeable solution thus far. If not, check if the size of the selected set *plus* a lower bound on the required number of rows to produce an encodeable set equals or exceeds the best encodeable solution obtained thus far. If so, return from this level of recursion.

2. Heuristically select a branching row.

3. Add this row to the selected set and recur for the sub-table resulting from deleting this row. Then, recur on deleting the row without adding it to the current set.

We are no longer concerned with covering the minterms in this branch-and-bound step, since all minterms have already been covered. We estimate the lower bound on the number of GPIs required to render the graph encodeable by finding the number of *disjoint* violations of the encodeability conditions of Theorem 3.5 and Theorem 4.1. In the sequel, we elaborate on disjoint violations.

If there are two cycles in the graph such that the edges in cycle 1 have different labels from all the edges in cycle 2 and no unselected GPI exists that contains both minterms corresponding to the labels of any pair of edges, then two GPIs are required to break both cycles. These two cycles are disjoint cycles. Similarly, assume we have two instances of directed paths between siblings of a conjunctive arc. If the two sets of edges in the two paths have disjoint sets of labels and no unselected GPI exists that covers the pair of minterms corresponding to any pair of edges in the two paths, then two GPIs are required to remove the two violations. We can have disjoint violations of Conditions 3 and 4 of Theorem 3.5 as well.

Disjoint violations of Condition 2 of Theorem 4.1 would have 2 state-tuples with dominance edge pairs that have different pairs of labels with the same GPI restriction as the restriction above. Similarly, one can have disjosint violations of Condition 3 of Theorem 4.1.

The heuristic selection of a GPI to add to the selected set at Step 2 is performed by selecting a GPI that covers a large number of minterms corresponding to the labels of edges that are involved in violations of the encodeability conditions.

7 Heuristics to Minimize the Number of Encoding Bits

Given a set of compatible input and output relations, in order to minimize the area of the PLA implementation, one wishes to construct an encoding satisfying all relations using a minimum number of bits.

Heuristics were proposed in [14] and [6] for encoding a set of input relations with a minimal number of bits. The heuristics of [14] were extended to include dominance relations in [13]. We propose the following procedure based on the procedures of [6] and [13].

1. Compact the set of input relations using techniques of [14] and [6]. Some input relations may be implied by others.

2. Represent the reduced set of input relations by a matrix, where each column corresponds to a symbolic value and each row to a constraint. Construct an encoding as the transpose of the matrix, i.e. each symbolic value/node receives as a code the column corresponding to the node in the original matrix. This encoding is guaranteed to satisfy the input relations [14].

3. Find the set of dominance relations between each pair of nodes that are not satisfied. No dominance relation could have been violated. Select a maximal disjoint set of pairwise dominance relations (By disjoint, we mean that the two nodes in the dominance relation are distinct from the nodes in the other dominance relation). Satisfy these relations by adding a single bit to the encoding. Do so till all dominance relations are satisfied.

4. Conjunctive relations have to satisfied for each bit in the encoding. If for a given conjunctive equality arc, a bit in the codes corresponding to the parent/siblings in the arc violates the

relation, it can only be because the bitwise OR of the siblings is a 0 and the parent is a 1 (This is because all the dominance relations have been satisfied). We try the choices of raising the bits in each possible subset of the siblings to a 1 (from a 0). At least one of these choices will not violate the dominance relations. However, an input relation may be violated and/or a dominance relation may no longer be satisfied.

5. For the input relations that are not satisfied, append a set of bits corresponding to the transpose of the compacted set of relations. Go to Step 3.

The procedure will converge since the set of relations is compatible.

8 Experimental Results

In this section, we present preliminary experimental results we have obtained on a set of examples. In our current implementation, generalized prime implicants are generated via the procedures of Section 3.4.1 and Secion 4.2.

The results obtained using the output encoding algorithm are given in Table 1. In the table, the number of inputs to the function (inp), the number of minterms in the original function (min), the number of symbolic values (val), the number of binary-valued outputs (out), the number of GPIs generated (gpi), the number of product terms in the minimized result (prod), the number of encoding bits (enc) and the CPU time in minutes required for GPI generation, covering and encoding on a microvax-III (CPU time) are given for each example. For example *ex5*, the covering problem could not be solved in less than a CPU-hour. For example *ex6* all the GPIs could not be generated due to memory limitations. However, examples *ex3* and *ex4* which have upto 20 symbolic values have been successfully encoded. An exhaustive search method is not feasible for these examples.

Results obtained using the state assignment algorithm are given in Table 2. The number of inputs (inp), states (sta), outputs (out) and edges (edg) are indicated for each FSM. Also, the number of GPIs generated (gpi), the number of product terms in a minimum encodeable result (prod), the number of encoding bits required (enc) and the CPU time in minutes on a microvax-III are given. Again for examples *fsm5* and *fsm6*, an exact solution could not be found. An exhaustive search method is only feasible for *fsm1*.

We believe that using the transformations of Section 6.2 prior to prime implicant generation will increase the size of the examples that can be handled, since a reduced prime implicant table can be

EX	inp	min	val	out	gpi	prod	enc	CPU time
ex1	2	4	4	1	6	3	2	0.1m
ex2	4	15	6	1	23	6	3	0.9m
ex3	6	44	16	2	194	14	6	10.4m
ex4	8	113	20	0	950	50	9	53.6m
ex5	10	213	20	1	8807	-	-	> 1h
ex6	12	410	32	0	> 9999	-	-	-

Table 1: Results Using Output Encoding Algorithm

EX	inp	sta	out	edg	gpi	prod	enc	CPU time
fsm1	1	3	1	6	13	3	3	0.05m
fsm2	1	8	1	16	91	2	3	4.1m
fsm3	7	16	4	118	1094	46	7	22.1m
fsm4	2	24	1	96	5810	23	12	43.7m
fsm5	8	20	6	107	> 9999	-	-	-
fsm6	7	19	2	170	> 9999	-	-	-

Table 2: Results Using State Assignment Algorithm

directly constructed. In the next section, we describe some CPU and memory-efficient heuristics based on the exact algorithms of Sections 3-5.

9 Computationally Efficient Heuristic Minimization Strategies

The exact algorithms described in Sections 3-5 may require inordinate amounts of memory or CPU time to run due to the following reasons:

1. The number of GPIs may be too large.

2. Checking for encodeability given a set of input, dominance and conjunctive relations can be accomplished in polynomial time by checking for the conditions of Theorem 4.1. However, we have a number of alternatives in choosing these relations. We conjecture that the problem of checking if an arbitrary set of GPIs can satisfy Eqn. 1 via some encoding is NP-complete, since the number of equations can be exponential and each equation represents multiple choices.

3. The classical covering problem is NP-complete and we have an additional branch-and-bound step in our covering problem.

The programs ESPRESSO-EXACT and McBOOLE appear to be capable of exactly minimizing most encoded FSMs. However, functions like those of Figure 12(b) and Figure 13(b) have huge numbers of PIs. This and having to continually check for encodeability are the reasons why the exact output encoding and state assignment algorithms fall short of the performance of exact two-level logic minimization algorithms. Many heuristics based on the exact procedures described here are now proposed. These heuristics may result in sub-optimal solutions but are computationally efficient.

During GPI generation as described in Section 3.4.1, one can discard (or not generate) GPIs with tags that contain more that $k \leq N - 1$ symbolic values, where N is tha total number of symbolic values. If $k = 1$, we have a disjoint minimization problem, equivalent to that of minimizing each of the symbolic value ON-sets separately. If $k = N - 1$, we have an exact output encoding algorithm. Reducing the number of symbolic values that can be contained in a GPI has two advantages. Firstly, the number of GPIs is reduced. Secondly, the encodeability check becomes easier because the constraints specified by Eqn. 1 are simpler.

Another strategy which can be used in conjunction with the above heuristic or in isolation is to define a *stronger* form of encodeability that is easier to check for. As long as the definition includes the conditions of Theorem 3.5 and Theorem 4.1, we will obtain an encodeable solution. The heuristic may miss an optimum solution because it may consider the solution not encodeable, when in reality it is encodeable. A stronger definition of encodeability would restrict the alternatives in satisfying a constraint given by Eqn. 1. Consider Eqn. 2. We have 7 possible choices in satisfying the constraint. If we restricted all relations to be dominance relations, we have 3 choices in satisfying Eqn. 2. Similarly, if we restricted all relations to be conjunctive, we have 4 alternatives. One can also spend a specified amount of time (or backtracks) checking for encodeability of a set of constraints and consider the selected set of GPIs to be not encodeable if a selection of edges representing a compatible set of relations has not been found within the prescribed limit.

A third heuristic approach is based on the iterative optimization strategy to two-level logic minimization, first used in MINI [10]. In the state assignment case for instance, one can begin with the given STT represented as GPIs and iteratively reduce, reshape and expand the GPIs while maintaining coverage of the minterms *and* encodeability. Since the iterative optimization approach has met with great success in the heuristic two-level logic minimization area, we feel that this approach holds the most promise.

10 Symbolic Don't Cares

Don't cares for binary-valued functions are simply represented and exploited in logic minimization. Functions with symbolic outputs may have associated don't care conditions with certain input combinations as well. These don't cares are called **symbolic don't cares.**

A symbolic don't care is defined on the set of symbolic values that the function can take. For instance, the cube 1010 in the function of Figure 14 is a symbolic don't care. A symbolic don't care may encompass all the symbolic values of the function or only a subset. Cube 1011 of Figure 14 is a don't care which can take on only a subset of the complete set of symbolic values.

One can produce an exact solution to an output encoding problem under an arbitrary symbolic don't care set as follows. Add the don't care minterms to the ON-sets of each of the symbolic values that the minterm can take. GPIs are generated as before. However, we may have a situation where two identical k-cubes have tags such that the first one's tag is a subset of the other. In this case the first k-cube cancels the second.

Given all the GPIs, the covering problem is solved as before. The minterms corresponding to the symbolic don't cares have to be covered as well and Eqn. 1 has to be satisfied for them, else

0000	out1
0011	out1
0001	out2
0100	out2
0101	out3
1000	out4
1010	out1/out2/out3/out4
1011	out1/out2

Figure 14: Symbolic Don't Cares

they may assert an invalid binary combination in the encoded cover. However, Eqn. 1 for these minterms has more choices, since a minterm effectively belongs to multiple symbolic value ON-sets (multiple v_ms in Eqn. 1). Any one of these constraints is to be satisfied. For example, we may have

$$out1 \cap out2 \bigcup out1 \cap out3 = out1 \text{ or } out2$$

for a symbolic don't care.

11 Conclusions

In this paper, we presented exact algorithms for the problems of output encoding, state assignment and four-level Boolean minimization.

The procedures described are much more efficient than a straightforward, exhaustive search procedure to solve these problems. We proposed a novel minimization procedure of prime implicant generation and covering that operates on symbolic outputs, rather than binary-valued outputs, for solving encoding problems.

Preliminary experimental results indicate that medium-sized problems can be solved exactly. Computationally efficient heuristic approaches based on the exact algorithms have been proposed. The efficiency and quality of these heuristic approaches is currently being evaluated.

12 Acknowledgements

The interesting discussions with Tony Ma, Robert Brayton and Alberto Sangiovanni-Vincentelli on state assignment are acknowledged. This research was supported in part by the Defense Advanced Research Projects Agency under contract N00014-87-K-0825.

References

[1] D. B. Armstrong. A programmed algorithm for assigning internal codes to sequential machines. In *IRE Transactions on Electron Computers*, pages 466–472, August 1962.

[2] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. Mis: a multiple-level logic optimization system. In *IEEE Transactions on CAD*, pages 1062–1081, November 1987.

[3] R. K. Brayton, G. D. Hachtel, Curt McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic Publishers, 1984.

[4] M. Dagenais, V. K. Agarwal, and N. Rumin. Mcboole: a procedure for exact boolean minimization. In *IEEE Transactions on CAD*, pages 229–237, January 1986.

[5] S. Devadas, H-K. T. Ma, A. R. Newton, and A. Sangiovanni-Vincentelli. Mustang: state assignment of finite state machines targeting multi-level logic implementations. In *IEEE Transactions on CAD*, pages 1290–1300, December 1988.

[6] S. Devadas, A. R. Wang, A. R. Newton, and A. Sangiovanni-Vincentelli. Boolean decomposition in multi-level logic optimization. In *Journal of Solid State Circuits*, April 1989.

[7] T. A. Dolotta. The coding of internal states of sequential machines. In *IEEE Transactions on Electronic Computers*, pages 549–562, October 1964.

[8] D. Bostick et. al. The boulder optimal logic design system. In *Int'l Conference on Computer-Aided Design*, November 1987.

[9] J. Hartmanis and R. E. Stearns. *Algebraic Structure Theory of Sequential Machines.* Prentice-Hall, Englewood Cliffs, N. J., 1966.

[10] S. J. Hong, R. G. Cain, and D. L. Ostapko. Mini: a heuristic approach for logic minimization. In *IBM journal of Research and Development*, pages 443–458, September 1974.

[11] E. Lawler. An approach to multi-level boolean minimization. In *Journal of the ACM*, 283-295 1964.

[12] E. J. McCluskey. Minimization of boolean functions. In *Bell Lab. Technical Journal*, pages 1417–1444, Bell Lab., November 1956.

[13] G. De Micheli. Symbolic design of combinational and sequential logic circuits implemented by two-level macros. In *IEEE Transactions on CAD*, pages 597–616, September 1986.

[14] G. De Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli. Optimal state assignment of finite state machines. In *IEEE Transactions on CAD*, pages 269–285, July 1985.

[15] R. Rudell and A. Sangiovanni-Vincentelli. Exact minimization of mutiple-valued functions for pla optimization. In *Proc. IEEE Int. Conf. on CAD (ICCAD)*, pages 352–355, 1986.

[16] R. Rudell and A. Sangiovanni-Vincentelli. Multiple-valued minimization for pla optimization. In *IEEE Transactions on CAD*, pages 727–751, September 1987.

[17] A. Saldanha and R. H. Katz. Pla optimization using output encoding. In *Int'l Conference on Computer-Aided Design*, pages 478–481, November 1988.

[18] T. Sasao. Pla decomposition. In *MCNC 1987 Logic Synthesis Workshop*, May 1987.

[19] P. Tison. Generalization of consensus theory and application to the minimization of boolean functions. In *IEEE Transactions on Computers*, pages 446–450, August 1967.

[20] H. C. Torng. An algorithm for finding secondary assignments of synchronous sequential circuits. In *IEEE Transactions on Computers*, pages 416–469, May 1968.

[21] W. Wolf, K. Keutzer, and J. Akella. A kernel finding state assignment algorithm for multi-level logic. In *Proc. of 25th Design Automation Conference*, pages 433–438, June 1988.