A FRAMEWORK FOR SATISFYING INPUT AND OUTPUT ENCODING CONSTRAINTS

by

Alexander Saldanha, Tiziano Villa, Robert K. Brayton,
and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M90/110

3 December 1990
(Revised November 19, 1991)

A FRAMEWORK FOR SATISFYING INPUT AND OUTPUT ENCODING CONSTRAINTS

by

Alexander Saldanha, Tiziano Villa, Robert K. Brayton,
and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M90/110

3 December 1990
(Revised November 19, 1991)

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A FRAMEWORK FOR SATISFYING INPUT AND OUTPUT ENCODING CONSTRAINTS

by

Alexander Saldanha, Tiziano Villa, Robert K. Brayton,
and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M90/110

3 December 1990
(Revised November 19, 1991)

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A Framework for Satisfying Input and Output Encoding Constraints

Alexander Saldanha* Tiziano Villa† Robert K. Brayton
Alberto L. Sangiovanni-Vincentelli

University of California - Berkeley CA

Abstract

Encoding is a significant step in the synthesis of digital circuits. Three relevant encoding problems are input, output and state encoding. Several algorithms have been proposed for their solutions that decompose the problem into symbolic minimization (yielding a set of constraints) and constraint satisfaction. At least two exact formulations of the input encoding constraint satisfaction problem exist. However, a more important use of encoding is in state assignment of finite state machines where both input and output encoding constraints must be satisfied to obtain the most effective implementations. This paper first proves that the encoding problem is NP-complete. Then a framework for the simultaneous satisfaction of input and output encoding constraints is developed. We describe an algorithm, polynomial in the number of symbols to be encoded, to check for the existence of a solution for a set of input and output constraints . An efficient algorithm that determines the minimum number of encoding bits required to satisfy all the given constraints is provided. We demonstrate how heuristic algorithms can be developed within the framework. Finally, the use of this framework in solving a variety of encoding problems with different cost functions is discussed. Results on standard benchmarks are given for both exact and heuristic algorithms.

1 Introduction

There are two stages of synthesis in which the circuit area may be minimized. The first is the conversion of a given register-transfer level description with symbolic inputs and/or outputs into a logic-level implementation by assigning binary codes to the symbolic variables leading to efficient area implementations. This problem is termed the encoding problem. The second is logic optimization that is performed on the implementation derived from the first stage. A wealth of contributions in the last decade advanced the theory of logic optimization for different styles of implementation [3]. A successful paradigm used in solving the encoding problem binds these two together by applying logic minimization on an unencoded specification followed by a transformation of the minimized symbolic representation to a compatible binary representation [7]. Thus, encoding also becomes a two phase process; first, obtaining a multi-valued

*Research support by SRC under contract 91-DC-008
†Research support by DARPA under contract JFBI90-073

minimized representation together with a set of constraints on the codes of the symbolic variables, and second, finding an encoding that satisfies the constraints. The constraints, if satisfied, are guaranteed to produce an encoded two-valued representation of the same cardinality as the multiple-valued minimized representation.

The encoding problem may be targeted for either two-level or multi-level logic implementations. Two-level implementations optimize the number of product-terms or the area of a PLA. Multi-level implementations optimize the number of literals of a technology-independent representation of the logic. Optimality may also be based on more complex criteria, such as performance and testability [18]. For two-level logic, an exact algorithm to implicitly generate all possible minimized symbolic covers of a given unencoded description together with the induced encoding constraints is provided in [9]. In the case of multi-level logic, constraints are generated by performing multiple-valued multi-level operations [13].

The various techniques for exact and heuristic encoding based on multiple-valued minimization of two-level and multi-level logic, reported in [7, 6, 9, 13, 4], produce one or more of four types of constraints.

The first type are *face-embedding* constraints generated by the multiple-valued input variables (*input* constraints). These constraints specify that a set of symbols is to be assigned to one *face* of a binary n-dimensional cube, without any other symbol sharing the same face. An input constraint involving symbols a, b and c is denoted by (a, b, c). An encoding satisfying (a, b, c) is given by $a = 11, b = 01, c = 00$. The vertex 10 cannot be assigned to any other symbol.

Two other types are *dominance* and *disjunctive* constraints generated by the multiple-valued output variables (*output* constraints). A dominance constraint (*covering*, denoted by $>$, e.g. $a > b$) requires that the code of a symbol (a) bit-wise cover the code of another symbol (b). A disjunctive constraint specifies that the code of a symbol (the *parent* symbol) is the bit-wise disjunction (*or-ing*, denoted by $\vee$, e.g. $a = b \vee c$) of the codes of two or more other symbols (the *children* symbols). Finally, the minimization procedure described in [9] produces disjunctive constraints with nested *conjunctive* terms. They require that the code of a symbol be the bit-wise disjunction (denoted by $\vee$) of the *conjunctions* (*and-ing*, denoted by $\wedge$) of the codes of two or more symbols. An example is: $(a \wedge b \wedge c) \vee (a \wedge d \wedge e) \vee (a \wedge f \wedge g) = a$.

An example containing input, dominance and disjunctive constraints is: (b, c), (c, d), (b, a), (a, d), $b > c$, $a > c$, $a = b \vee d$. An encoding satisfying all constraints exists and the minimum code length is two. A solution is $a = 11, b = 01, c = 00, d = 10$.

A face embedding constraint is used to express several input terms of an output function by a single term. This results in a smaller cover for the output function. A dominance constraint is used to increase the don't care set associated with an output function, whenever this leads to a smaller implementation. A dominance constraint is useful only in conjunction with a related face embedding constraint, when encoding both input and output variables. A disjunctive constraint is used to decrease the size of the cover of an output function. This occurs if the input parts of the output corresponding to the parent symbol are contained in each of the input parts of the outputs corresponding to the children symbols. These constraints arise with both face embedding and dominance constraints, when encoding both input and output variables.

Disjunctive constraints with nested conjunctive terms are used in the exact encoding technique described in [9] to ensure that if a set of *generalized prime implicants* is chosen as a solution, each minterm asserts the same output combination as it would have in the original cover. Each generalized prime implicant asserts the conjunction of the codes of a group of symbols in its output part. The output for a minterm in a two-level implementation is then the disjunction of all the outputs asserted by the generalized prime

implicants that cover the minterm.

The origin and the effect of these constraints on the quality of the minimized result is described in [7, 6, 9, 13].

In this paper, we focus on the following problems. Given a set of encoding constraints returned by a symbolic minimization procedure:

P-1: Determine whether the constraints are satisfiable.

P-2: Determine the binary codes that use a minimum number of bits and satisfy all the constraints.

P-3: Minimize a cost function of the constraints that are satisfiable using a fixed number of code bits.

Previous work on encoding constraint satisfaction has concentrated on solving input constraints, alone. Exact algorithms and efficient heuristics for solving problems P-2 and P-3 are reported in [24, 25]. An approximate solution to P-3 based on a theory of intersecting cubes is described in [20, 10]. Problem P-1 is trivial for only input constraints. A solution to P-1 and a heuristic algorithm to solve P-3 when both input and output dominance constraints occur are provided in [6]. A solution to P-1 when both input and output constraints (including disjunctive constraints) are present is attempted in [9], but we will demonstrate that the algorithm is incomplete. To date, to the best of our knowledge, no effective algorithms exist for solving all three problems when all types of constraints occur.

In this paper we propose a framework and efficient algorithms to solve P-1, P-2 and P-3 for input and output encoding constraints. A polynomial time algorithm to answer P-1 and exact and heuristic algorithms to solve P-2 and P-3 are provided. We solve P-3 with different cost functions, such as the number of constraints satisfied and the number of cubes or literals required in the encoded implementation. These algorithms also handle encoding don't cares [6, 13] and can be easily extended to other types of constraints. We also prove the NP-completeness of problems P-2 and P-3. This result has not been shown previously, though it has been conjectured [6].

The approach used here is based on a formulation provided in [25] and related to the state assignment technique employed by Tracey in 1966 [23]. We first demonstrate the difficulty of the encoding problem by proving it NP-complete in Section 2. Section 3 provides some definitions. In Section 4 an abstraction of the problem is presented. In Section 5 we describe a new algorithm to satisfy input constraints alone. This is extended to handle input and output constraints in Section 6. In Section 6.2 we first show that a previous algorithm for checking the satisfiability of encoding constraints [9] is incomplete and we provide an alternate correct algorithm. A heuristic algorithm is briefly sketched in Section 7. Extensions of the framework to handle various types of constraints and cost functions are discussed in Section 8. Substantial experimental results are provided in Sections 9 and conclusions are described in Section 10.

This paper is an extended version of [19].

2 Statement and complexity of the encoding problems

Three types of problems related to the satisfaction of input and output constraints were introduced informally in the Section 1. They are:

P-1: Determine whether the constraints are satisfiable.

P-2: Determine the binary codes that use a minimum number of bits and satisfy all the constraints.

P-3: Minimize a cost function of the constraints that are satisfiable by a fixed number of code bits.

Problem P-1 can be solved by a polynomial algorithm, as will be shown later. In this section we formally state P-2 both as a decision and an optimization problem and show that the decision (optimization) version with input constraints alone is NP-complete (NP-hard).

A few preliminary definitions are required. An *n-dimensional hypercube* (or n-cube) is a graph of 2^n vertices labeled uniquely by the integers from 0 and $2^n - 1$. An edge joins two vertices whose binary representations of their integer labels differ by exactly one bit. The minimum k-cube that contains a given subset of vertices of a n-cube ($k \leq n$) is the k-face (or smallest face) spanned by the given vertices.

Decision version of P-2:

Instance: Set of input and output constraints defined on a set of symbols, S, and a positive integer k.

Question: Is there a function f from S to the vertices of a k-cube such that:

1. Symbols in the same input constraint are mapped to vertices spanning a face that does not contain the image of any other symbol.
2. The binary labels of the images of the symbols satisfy the output constraints?

Optimization version of P-2:

Instance: Set of input and output constraints defined on a set of symbols, S.

Objective: Find the minimum k-cube and a function f from S to the vertices of the k-cube such that:

1. Symbols in the same input constraint are mapped to vertices spanning a face that does not contain the image of any other symbol;
2. The binary labels of the images of the symbols satisfy the output constraints.

Answering the decision version for different dimensions repeatedly solves the optimization problem (with a polynomial number of calls to the decision procedure) and, of course, solving the optimization problem answers the decision version for all dimensions. Clearly, by assigning a weight of *1* to each constraint, P-2 can be seen as a special case of P-3. Hence, P-3 is no easier that P-2.

The decision version of P-2 with input constraints alone is defined as *face hypercube embedding*. To prove that face hypercube embedding is NP-complete a few more preliminaries are needed.

A given graph $G = (V, E)$ is a subgraph of an n-cube if there is a function mapping vertices of G into vertices of the n-cube that preserves the adjacency relations. G can be embedded in an n-cube if G is a subgraph of the n-cube. The problem of deciding whether a given graph is embeddable into an arbitrary dimension hypercube has been shown to be NP-complete [12]. It has also been proved that even the problem of deciding whether a graph can be embedded into a fixed-size hypercube is NP-complete [5]. The proof in [5] actually shows that the problem of determining whether a graph of 2^k nodes can be embedded in a k-cube is NP-complete. This result can be used to prove that face hypercube embedding is NP-complete.

Theorem 2.1 *Face hypercube embedding is NP-complete.*

Proof Face hypercube embedding is in NP. The assignment of symbols to vertices can be non-deterministically checked in polynomial time.

Suppose k is the dimension of the hypercube into which the face constraints composed of simbols from set S must be embedded. Let us restrict face hypercube embedding to the instances where the symbols involved in the face constraints are 2^k and each face constraint involves only two symbols. For these instances it is possible to define a graph $G(V, E)$ induced by the face constraints. The set of nodes V is in correspondence with the symbols in S and there is an edge between two nodes when the two corresponding symbols are in the same face constraint. The set of face constraints can be embedded into a k-cube if and only if the companion graph is a subgraph of a k-cube. Notice that in this case the concept of face embedding reduces to the familiar notion of graph adjacency. The problem of determining whether a graph of 2^k nodes is a subgraph of a k-cube has been proved to be NP-complete by reduction from 3-partition [5]. Therefore the problem of determining whether for 2^k symbols a set of face constraints each with exactly two symbols can be embedded into a k-cube is NP-complete. But this is a restricted version of face hypercube embedding and hence the latter too is NP-complete. ■

3 Definitions

Definition 3.1 *An* **encoding-dichotomy** *is a 2-block partition of symbols to be encoded. The symbols in the left block are associated with the bit* 0 *while those in the right block are associated with the bit* 1. *If an encoding-dichotomy is used in generating an encoding, then one code bit of the symbols in the left block is assigned* 0 *while the same code bit is assigned* 1 *for the symbols in the right block.*

For example, $(s_0s_1; s_2s_3)$ is an encoding-dichotomy in which s_0 and s_1 are associated with the bit *0* and s_2 and s_3 are associated with the bit *1*. This definition differs from that of a *dichotomy*, defined in [23, 25], where a dichotomy allows the left block to assume either the encoding bit *0* or *1*. As will be shown later, this extension is useful in describing output encoding constraints.

Definition 3.2 *Two encoding-dichotomies* d_1 *and* d_2 *are* **compatible** *if the left (right) block of* d_1 *is disjoint from the right (left) block of* d_2. *Otherwise,* d_1 *and* d_2 *are incompatible.*

Note again that this definition differs from the definition of compatibility described in [23, 25].

Definition 3.3 *The* **union** *of two compatible encoding-dichotomies,* d_1 *and* d_2, *is the encoding-dichotomy whose left and right blocks are the union of the left and right blocks of* d_1 *and* d_2, *respectively.*

The union operation is not defined for incompatible encoding-dichotomies.

Definition 3.4 *An encoding-dichotomy* d_1 **covers** *an encoding-dichotomy* d_2 *if the left and right blocks of* d_2 *are subsets respectively either of the left and right blocks, or of the right and left blocks of* d_1.

For example, $(s_0; s_1s_2)$ is covered by $(s_0s_3; s_1s_2s_4)$ and $(s_1s_2s_3; s_0)$, but not by $(s_0s_1; s_2)$.

Definition 3.5 *A* **prime encoding-dichotomy** *of a given set of encoding-dichotomies is one that is incompatible with all encoding-dichotomies not covered by it.*

A set of prime encoding-dichotomies generates an encoding as follows. Each prime generates one column of the encoding, with symbols in the left (right) block assigned a *0* (*1*) in that column. For example, given the prime encoding-dichotomies $(s_0s_1; s_2s_3)$ and $(s_0s_3; s_1s_2)$, the unique encoding derived is $s_0 = 00, s_1 = 01, s_2 = 11$ and $s_3 = 10$.

Definition 3.6 *An encoding-dichotomy* **violates** *an output constraint if the encoding bit generated for the symbols in the encoding-dichotomy does not satisfy the bit-wise requirement for the output constraint. A* **valid encoding-dichotomy** *is one that does not violate any output constraint.*

For example, the encoding-dichotomy $(s_0; s_1s_2)$ violates the constraint $s_0 > s_1$, since s_0 is assigned bit *0* whereas s_1 is assigned bit *1* by this encoding-dichotomy. Hence, s_0 does not cover s_1 in this bit. The encoding-dichotomy $(s_0s_1; s_2)$ does not violate this constraint.

4 Abstraction of the problem

The combinatorial problems of encoding constraints satisfaction can be abstracted as binate covering problems.

We first recall the definition of binate covering for the benefit of the following discussion. Suppose that a Boolean expression T is given in product-of-sums form. Associate a cost w_j to each literal α_j of a set of t literals, and say that a given assignment has cost:

$$\sum_{j=1}^{t} w_j \alpha_j.$$

Consider the problem of finding an assignment of minimum cost that satisfies T. Writing T as an array of cubes (that form a matrix with coefficients from the set $\{0, 1, 2\}$), the problem can be stated as a *binate covering* problem [16]: find a subset C of columns of minimum cost, such that for every row r_i, either

1. $\exists j$ such that $t_{ij} = 1$ and $c_j \in C$, or
2. $\exists j$ such that $t_{ij} = 0$ and $c_j \notin C$.

In the unate covering problem, the coefficients of T are only 1's and 2's and only the first condition must hold.

To illustrate the opening statement, suppose that the symbols a, b, c and the three constraints $(a, b), b > c, b = a \vee c$ are given. All possible encodings can be considered by enumerating all the patterns that yield one bit of encoding to each symbol. Each such bit pattern is termed an *encoding column*. Consider all possible column encodings for the example, $c_1 = 001, c_2 = 010, c_3 = 011, c_4 = 100, c_5 = 101, c_6 = 110$, where the order of symbols in a column is a, b, c[1]. For each face constraint consider the encoding dichotomies that have the symbols of the face constraint in one block, and have one of the remaining symbols in the other block [25]. In the given example, this is $ab; c$. Add the encoding dichotomies expressing the uniqueness of the codes In the example, these are $a; b$, $a; c$, $b; c$. Build a table whose columns are the encoding columns and whose rows are the encoding dichotomies. Put a 1 in entry (i, j) if column j covers row i. For each output constraint, add one row for each encoding column that cannot be chosen

[1] Bit patterns 000 and 111 are excluded because they carry no useful information.

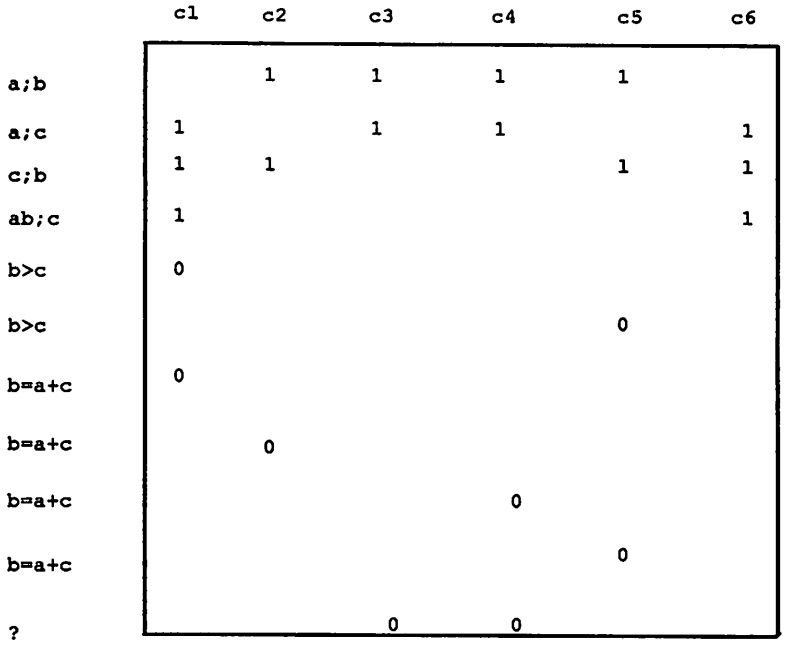

Figure 1: Satisfaction of constraints as binate covering

if that output constraint must be satisfied and put a 0 in the corresponding entry. In the example, $a > b$ yields two rows, one has a 0 in the position of column c_1, the other has a 0 in the position of column c_5. One could imagine more complex types of constraints that add rows carrying two or more 0s and no 1 to denote that not all the corresponding columns can be selected at the same time to have the row covered. The binate table of the given example is shown in Figure 1. A minimum column cover of the given rows gives a minimum set of encoding columns that satisfy all given constraints. In the general case this requires the solution of a binate covering problem. If only face constraints are present, the problem reduces to a unate covering problem [23].

In summary, solving problem P-1 can be reduced to deciding whether a solution to a binate covering problem exists, solving P-2 can be reduced to solving a binate covering problem, and solving P-3 can be reduced to fixing the number of columns that can be used in a solution of a binate covering problem. This may sound hopeless, since binate covering is a hard problem. In the next sections we will show that problem P-1 can be decided by a polynomial time algorithm and that an efficient exact algorithm can be devised to solve P-2 on cases of practical interest. When an exact solution is not feasible, as in the case of P-3, a heuristic algorithm handling different cost functions is described.

5 Input constraint satisfaction

We first present a new algorithm for satisfying input encoding constraints that, compared to previous approaches [24, 25], significantly improves the efficiency of the input encoding process.

The encoding constraint satisfaction problem is a three step process. The first is the generation of the encoding-dichotomies that represent the face embedding constraints [25]. Each face embedding constraint generates several encoding-dichotomies. The symbols that are to be on a face are placed in one block of each encoding-dichotomy representing that constraint, while the other block contains one

of the symbols not in the face. Thus, for n symbols $s_1, s_2, ..., s_n$ and a face embedding constraint that requires the l symbols $s_1, s_2, .., s_l$ to be in one face, we generate 2 $(n - l)$ encoding-dichotomies each with the symbols $s_1, s_2, ..., s_l$ in one block (either left or right) and exactly one of the remaining $n - l$ symbols in the other block.

These encoding-dichotomies exactly capture the face embedding constraints. We also require that each symbol gets a distinct code. This is represented by an encoding-dichotomy with one symbol in each block. We need to add only those uniqueness constraints that are not covered by the encoding-dichotomies generated from the face-embedding constraints. When there are n symbols and no encoding constraints, the number of uniqueness constraints is $n^2 - n$; these would generate an exponential number $(2^n - 2)$ of prime encoding-dichotomies.

The second step of encoding is the generation of prime encoding-dichotomies from the encoding-dichotomies. [23] describes an approach similar to the process of *iterated consensus* for prime generation in two-level logic minimization [2]. However, the number of iterations required to generate all the prime encoding-dichotomies may be formidable even for small problems. Using this approach, several different compatible mergings often yield the same prime encoding-dichotomy. This results in a substantial waste of computation time [25]. In Section 5.1, we describe a method of generating all prime encoding-dichotomies and demonstrate its effectiveness in determining an exact solution.

The final step of encoding is to obtain a cover of the initial encoding-dichotomies using a minimum number of primes. This is a classical unate covering problem and efficient branch and bound techniques, both for exact and heuristic solutions, are well known; hence we do not discuss this further in the paper.

5.1 Efficient generation of prime encoding-dichotomies

By definition, each prime encoding-dichotomy is a *maximal compatible* of the encoding-dichotomies since it is not compatible with any encoding-dichotomy that it does not cover. As in [15], an incompatibility between two encoding-dichotomies, each represented by the literals a and b respectively, is written as $(a + b)$. When the product of the sum terms representing all the pairwise incompatibilities is written as an irredundant sum-of-products expression, a maximal compatible is generated as the union of those encoding-dichotomies whose literals are missing in any product term [15]. For example, assume that we wish to find the maximal compatibles for five encoding-dichotomies, a, b, c, d, e. Assume the incompatibilities are $(a + b)(a + c)(b + c)(c + d)(d + e)$. Then the equivalent irredundant sum-of-products expression is $acd + ace + bcd + bce$. The four encoding-primes are then formed by the unions of the missing literals, $\{b, e\}, \{b, d\}, \{a, e\}, \{a, d\}$.

The problem is how to efficiently derive the equivalent sum-of-products expression from the product-of-sums expression representing the incompatibilities. In the past, this has been performed using an approach based on Shannon decomposition [15]. The complexity of performing the recursive *splitting* and *merging* is exponential since a binary tree is being constructed. It is not practical for any reasonably sized encoding problem. We describe an algorithm that can generate all the encoding-primes, but only uses a linear number of recursive *splitting* and *merging* operations.

Since each sum term in the product-of-incompatibilities expression has exactly two literals, we adapt the classical 2-SAT algorithm to this problem[2]. The algorithm is described in pseudo-code in Figure 2. Given a product-of-sums expression, a splitting variable, x, is chosen. The product of all sum terms containing x, call it *x_expr*, when simplified consists of two terms, the first is x alone and the second is

[2]It is well known that 2-SAT is solvable in linear time in the number of clauses.

the product of all the other variables in *x_expr*. A recursive call is applied to the product of the sum-terms in the initial expression that do not contain x, called *reduced_expr*. The two product terms, *x_expr* and *cs(reduced_expr)*, are multiplied and single cube-containment is used to obtain the minimum sum-of-products expression[3].

This algorithm replaces an exponential number (in the number of encoding-dichotomies) of recursive calls by a linear number. Of course, the worst case complexity is linear in the final number of encoding-primes, which may be exponential. In practice this does not happen, especially with highly constrained problems. As already mentioned, the number of encoding-primes in an unconstrained problem (no face embedding constraints) is exponential in number due to the uniqueness constraints. Each face embedding constraint serves to reduce these encoding-primes.

The example in Figure 3 illustrates the complete input encoding process. A set of input constraints is shown and the corresponding initial encoding dichotomies are derived. The maximal compatibles are generated by a procedure that recurs on the splitting variable. Variable 0 is chosen as first splitting variable. The procedure returns the minimal product of the following two expressions: the first is the product of all sum terms containing 0 (in this case simplified into 0 and 2345678) and the second is the result of the recursive call of the procedure on the sum terms that do not contain 0. By minimal product it is meant that the two expressions, when available after a series of recursive calls, are multiplied out and then single cube-containment is performed on them. Once the maximal compatibles are found, the prime encoding-dichotomies are easily obtained and a standard unate covering routine produces a minimum subset of primes that cover all given initial encoding-dichotomies. Notice that to simplify the example we have forced the symbol s_1 to always be in a right block. This reduces the number of prime encoding-dichotomies but does not affect the solution to the input encoding problem[4].

6 Input and output constraint satisfaction

6.1 Output encoding constraints

A dominance constraint $a > b$, requires that the encoding for a bit-wise covers the encoding for b. This means that any prime encoding-dichotomy chosen in the final cover cannot have a in the left block while b is in the right block. Hence, any encoding-dichotomy that has this property may be deleted from consideration.

A disjunctive constraint $a = b \vee c$, implies that the encoding for symbol a must be the same as the bit-wise *or* of the encodings of b and c. This means that any prime encoding-dichotomy in a feasible solution must have at least one of b and c appear in the same block as a. Any encoding-dichotomy that does not possess this property may be deleted. This property is easily extended to the case where the disjunctive constraint involves more than two symbols or has nested conjunctive constraints.

A preliminary algorithm follows from the discussion above. In the first step, the encoding-dichotomies corresponding to the input constraints are generated. Next the prime encoding-dichotomies are generated using the algorithm described in Section 5.1; those that violate any of the dominance or disjunctive constraints are eliminated. Finally, the remaining primes are used in selecting a minimum cover of all the

[3]Single cube-containment can be used to find the minimum expression since the function is unate [2].

[4]In general, this symmetry cannot be exploited when there are both input and output constraints.

```
/* Given pairwise incompatibilities among a list of encoding-dichotomies
as a product-of-sums expression generate all encoding-prime dichotomies.
Each sum term has two literals and there are n variables,
each corresponding to a distinct initial encoding-dichotomy. */

/* Convert 2-CNF to sum-of-products expression */
O(n) recursive calls */
procedure cs (expr) {
    x = splitting variable
    C = all sum terms with variable x
    reduced_expr = expr without the sum-terms in C
    x_expr = sum-of-product expression of C
    return (ps (x_expr, cs(reduced_expr)))
}

/* Obtain the product of two expressions.
expr1 has 2 terms, where the first term is a single variable */
procedure ps (expr1, expr2) {
    product_expr = product of expr1 and expr2
    result_expr = single_cube_containment_minimal (product_expr)
    return (result_expr)
}

procedure prime_dichotomy_generate (expr) {
    result = cs (expr)
    foreach (term T in result)
        missing = list of variables not in T
        new_prime_dichotomy = union of encoding-dichotomies
            corresponding to missing
        add new_prime_dichotomy to prime_list
    return (prime_list)
}
```

Figure 2: Efficient generation of prime encoding-dichotomies

Constraints	(s_0, s_2, s_4)	(s_0, s_1, s_4)	(s_1, s_2, s_3)	(s_1, s_3, s_4)
Initial encoding – dichotomies	$1:(s_1; s_0s_2s_4)$	$2:(s_3; s_0s_2s_4)$	$3:(s_3; s_0s_1s_4)$	$4:(s_2; s_0s_1s_4)$
$5:(s_0; s_1s_2s_3)$	$6:(s_4; s_1s_2s_3)$			
$7:(s_0; s_1s_3s_4)$	$8:(s_2; s_1s_3s_4)$			

Deriving maximal compatibles (prime encoding – dichotomies)

$cs(\ (0+2)(0+3)(0+4)(0+5)(0+6)(0+7)(0+8)(1+2)(1+3)(1+4)(1+5)(1+6)(1+7)(1+8)(2+4)(2+5)(2+6)(2+7)(2+8)(3+5)(3+6)(3+7)(3+8)(4+5)(4+6)(4+7)(5+8)(6+7)(6+8)\ )$

$ps(\ (0+2345678), cs(\ (1+2)(1+3)(1+4)(1+5)(1+6)(1+7)(1+8)(2+4)(2+5)(2+6)(2+7)(2+8)(3+5)(3+6)(3+7)(3+8)(4+5)(4+6)(4+7)(5+8)(6+7)(6+8)\ )\ )$

$ps(\ (0+2345678), ps(\ (1+2345678), cs(\ (2+4)(2+5)(2+6)(2+7)(2+8)(3+5)(3+6)(3+7)(3+8)(4+5)(4+6)(4+7)(5+8)(6+7)(6+8)\ )\ )\ )$

$ps(\ (0+2345678), ps(\ (1+2345678), ps(\ (2+45678), cs(\ (3+5)(3+6)(3+7)(3+8)(4+5)(4+6)(4+7)(5+8)(6+7)(6+8)\ )\ )\ )\ )\ )$

$ps(\ (0+2345678), ps(\ (1+2345678), ps(\ (2+45678), ps(\ (4+567), cs(\ (3+5)(3+6)(3+7)(3+8)(5+8)(6+7)(6+8)\ )\ )\ )\ )\ )$

$ps(\ (0+2345678), ps(\ (1+2345678), ps(\ (2+45678), ps(\ (4+567), ps(\ (3+5678), cs(\ (5+8)(6+7)(6+8)\ )\ )\ )\ )\ )\ )$

$ps(\ (0+2345678), ps(\ (1+2345678), ps(\ (2+45678), ps(\ (4+567), ps(\ (3+5678), ps(\ (6+78), cs(\ (5+8)\ )\ )\ )\ )\ )\ )\ )$

$ps(\ (0+2345678), ps(\ (1+2345678), ps(\ (2+45678), ps(\ (4+567), ps(\ (3+5678), (56+68+78)\ )\ )\ )\ )\ )$

$ps(\ (0+2345678), ps(\ (1+2345678), ps(\ (2+45678), ps(\ (4+567), (356+368+378+5678)\ )\ )\ )\ )$

$ps(\ (0+2345678), ps(\ (1+2345678), ps(\ (2+45678), (3456+3468+3478+3567+5678)\ )\ )\ )$

$ps(\ (0+2345678), ps(\ (1+2345678), (23456+23468+23478+23567+25678+45678)\ )\ )$

$ps(\ (0+2345678), (123456+123468+123478+123567+125678+145678+2345678)\ )$

$(0123456+0123468+0123478+0123567+0125678+0145678+2345678)$

Maximal compatible sets	$\{7,8\}$	$\{5,7\}$	$\{5,6\}$	$\{4,8\}$
	$\{3,4\}$	$\{2,3\}$	$\{0,1\}$	
Prime encoding – dichotomies	$(s_0s_2s_4; s_1s_3)$	$(s_3; s_0s_1s_2s_4)$	$(s_2s_3; s_0s_1s_4)$	$(s_2; s_0s_1s_3s_4)$
	$(s_0s_4; s_1s_2s_3)$	$(s_0s_2; s_1s_3s_4)$	$(s_0; s_1s_2s_3s_4)$	
Minimum cover	$(s_0s_2s_4; s_1s_3)$	$(s_2s_3; s_0s_1s_4)$	$(s_0s_4; s_1s_2s_3)$	$(s_0s_2; s_1s_3s_4)$

Figure 3: Input encoding example

initial encoding-dichotomies representing the input constraints. If there is at least one initial encoding-dichotomy that cannot be covered, then there is no solution.

This procedure may be used to answer two questions. The first is whether a feasible encoding exists for a set of input and output constraints. The second is to find the minimum length encoding satisfying the constraints, if it exists. An obvious drawback of this method is that many prime encoding-dichotomies may be generated but later deleted since they violate output constraints. Using the framework discussed above, we present an efficient algorithm that avoids the generation of useless prime encoding-dichotomies.

6.2 Input and output constraint satisfaction

We motivate the constraint satisfaction procedure using the example in Figure 4 [5]. Given the input and output constraints, 26 initial encoding-dichotomies are obtained. Consider the initial encoding-dichotomies $(s_0; s_1s_5)$ and $(s_1s_5; s_0)$ that are generated from the face embedding constraint (s_1, s_5). Since $s_0 > s_1$, the encoding-dichotomy $(s_0; s_1s_5)$ is not allowed and is deleted from consideration. The encoding-dichotomy $(s_1s_5; s_0)$ is valid and will be used in a feasible encoding. Consider the encoding-dichotomy $(s_1; s_2s_5)$. If this encoding-dichotomy is to be expanded to a valid prime encoding-dichotomy, symbol s_0 is forced to be in the right block, since $s_0 > s_2$. Also, since $s_1 > s_3$, s_3 must be in the left block and since $s_4 > s_5$, s_4 is forced into the right block. Thus, all valid encoding-dichotomies covering this initial encoding dichotomy must cover the "raised" dichotomy $(s_1s_3; s_0s_2s_4s_5)$. Similarly, we obtain the six raised encoding-dichotomies shown. On generating the prime encoding-dichotomies from these raised encoding-dichotomies, we obtain five primes.

Definition 6.1 *An encoding-dichotomy is* **raised** *by adding symbols into either its left or right block as implied by the output constraints.*

For example, the encoding-dichotomy $(s_0; s_1s_2)$ may be raised to the encoding-dichotomy $(s_0s_3; s_1s_2)$. We use this definition in the context of output constraints to restrict the number of prime dichotomies to be considered in an exact solution.

Definition 6.2 *An encoding-dichotomy is said to be* **maximally raised** *if no further symbols can be added into either the left or right block by the output constraints.*

The procedure *raise-dichotomy* in Figures 5 and 6 describes an algorithm that maximally raises an encoding-dichotomy with respect to a set of output constraints.

When the problem is to determine if a set of constraints is satisfiable, we do not have to generate the prime encoding-dichotomies. Instead we use the set of maximally raised valid encoding-dichotomies, which are far fewer in number than the prime encoding-dichotomies. As shown below, this smaller set of dichotomies is also sufficient to solve all the problems of interest.

We merely check if all the initial encoding-dichotomies are covered by the maximally raised and valid encoding-dichotomies. This condition is formally stated below.

[5] An algorithm was provided in [9] to check for satisfiability of input and output constraints, by checking for conditions that ensure non-conflicting input and output constraints. However, for the example in Figure 4, for which no feasible encoding exists, the algorithm in [9] states that the constraints are satisfiable. Except for the approach described in this paper, we know of no efficient and correct algorithms for checking satisfiability of mixed input and output constraints.

Face embedding constraints :

(s_1, s_5)	(s_2, s_5)	(s_4, s_5)

Dominance constraints :

$s_0 > s_1$	$s_0 > s_2$	$s_0 > s_3$
$s_0 > s_5$	$s_1 > s_3$	$s_2 > s_3$
$s_4 > s_5$	$s_5 > s_2$	$s_5 > s_3$

Disjunctive constraints :

$s_0 = s_1 \vee s_2$

Initial encoding – dichotomies :

$(s_0; s_1s_5)$	$(s_1s_5; s_0)$	$(s_0; s_2s_5)$
$(s_2s_5; s_0)$	$(s_0; s_4s_5)$	$(s_4s_5; s_0)$
$(s_1; s_2s_5)$	$(s_2s_5; s_1)$	$(s_1; s_4s_5)$
$(s_4s_5; s_1)$	$(s_2; s_1s_5)$	$(s_1s_5; s_2)$
$(s_2; s_4s_5)$	$(s_4s_5; s_2)$	$(s_3; s_1s_5)$
$(s_1s_5; s_3)$	$(s_3; s_2s_5)$	$(s_2s_5; s_3)$
$(s_3; s_4s_5)$	$(s_4s_5; s_3)$	$(s_4; s_1s_5)$
$(s_1s_5; s_4)$	$(s_4; s_2s_5)$	$(s_2s_5; s_4)$
$(s_0; s_3)$	$(s_3; s_0)$	

Raised encoding – dichotomies :

$(s_1s_3; s_0s_2s_4s_5)$	$(s_2s_3; s_0s_1s_4s_5)$	$(s_2s_3s_4s_5; s_0s_1)$
$(s_0s_1s_2s_3s_5; s_4)$	$(s_2s_3s_5; s_0s_1)$	
$(s_2s_3s_5; s_4)$		

Uncovered initial encoding – dichotomies :

$(s_0; s_1s_5)$

$(s_1s_5; s_0)$

Figure 4: Example of feasibility check with input and output constraints

Theorem 6.1 *Given a set of input and output constraints, let I be the set of initial encoding-dichotomies generated from the input constraints, including all uniqueness constraints. Let each valid encoding-dichotomy in I be maximally raised to obtain a set of valid encoding-dichotomies D. An encoding-dichotomy that becomes invalid on raising is deleted. The input and output constraints are satisfiable if and only if each $i \in I$ is covered by some $d \in D$.*

Proof

If Part Consider some valid maximally raised encoding-dichotomy $d = (L_1; R_1)$, where L_1 and R_1 are disjoint subsets of the symbols to be encoded. Consider a symbol $s \notin d$. There are no output constraints that either require any of the symbols in L_1 to cover s, or s to cover any of the symbols in R_1. Otherwise d is not raised maximally. So s may be added to either block of d. Add all symbols $S = \{s : s \notin d\}$, to the right block of d. There may be output constraints among the symbols in S, but these are satisfied since all the symbols in S are inserted into the right block. Hence, a valid encoding exists by merely deriving the codes from all of the raised encoding-dichotomies that cover all encoding-dichotomies in I.

Only If Part Assume some encoding-dichotomy $i \in I$ is not covered by any of the encoding-dichotomies in D. Then by merging compatible dichotomies in D, i can never be covered. Hence, no feasible solution exists in this case. ■

Theorem 6.1 can be extended to the case of disjunctions of nested conjunctions (*extended disjunctive constraints*) that are generated by the minimization procedure given in [9]. Each such constraint causes the correct output symbol to be produced for each input in an encoded cover. The form of an extended disjunctive constraint for a minterm m, asserting output s_m and a set of selected GPIs is:

$$(\bigvee_{g \in G} \bigwedge_{s \in O_g} s) = s_m$$

where G is the subset of the selected GPIs that cover m, and O_g is the set of output symbols that the GPI g asserts. By construction, [9], $\forall g, s_m \in O_g$; applying the distributive law, one gets:

$$s_m \wedge (\bigvee_{g \in G} \bigwedge_{s \in O_g - s_m} s) = s_m$$

This implies that,

$$(\bigvee_{g \in G} \bigwedge_{s \in O_g - s_m} s) \geq s_m$$

An example is $(a \wedge b \wedge c) \vee (a \wedge d \wedge e) \vee (a \wedge f \wedge g) = a$, that becomes $a \wedge ((b \wedge c) \vee (d \wedge e) \vee (f \wedge g)) = a$, and gives $(b \wedge c) \vee (d \wedge e) \vee (f \wedge g) >= a$. This constraint requires that for each bit of an encoding where a is assigned bit *1*, both the states in at least one of the three pairs (b, c), (d, e), and (f, g) must be assigned bit *1*.

The proof of Theorem 6.1 holds *verbatim*. This problem was conjectured to be NP-complete in [9], but in fact is solved in polynomial time here.

An algorithm to check for the satisfiability of input and output constraints is shown in Figures 5 and 6. Since the raising of each encoding-dichotomy is performed in time linear in the number of symbols times the number of initial encoding-dichotomies, the algorithm complexity is polynomial in the number of symbols and constraints.

```
/* S is the set of symbols to be encoded */
procedure remove_invalid_dichotomies (D, constraints) {
    /* to handle dominance and disjunctive constraints */
    foreach (encoding-dichotomy d ∈ D)
        foreach (pair of symbols s, m ∈ S)
            if (s > m & s in left block & m in right block)
                delete d
        foreach (disjunctive constraint)
            if (parent in left block & all children not in left block)
                delete d
            if (parent in right block & all children in left block)
                delete d

    /* to handle extended disjunctive constraints */
    foreach (extended disjunctive constraint)
        if (parent in right block &
            one child of each conjunction in left block)
            delete d }

/* d is a valid encoding-dichotomy */
procedure raise_dichotomy (d, constraints) {
    do {
    /* to handle dominance and disjunctive constraints */
    foreach (symbol s ∈ S)
        if (s in left block & s > m)
            insert m into left block of d
        if (s in the right block & m > s)
            insert m into right block of d
    foreach (parent symbol s in a disjunctive constraint)
        if (all children in the left block)
            insert s into left block
        if (all but one children are in left block &
                s is in the right block)
            insert last child into right block

    /* to handle extended disjunctive constraints */
    foreach (parent s in an extended disjunctive constraint e)
            if (one child of each conjunction in the left block)
                insert s into left block
            if (one child of all but one conjunction are in left block &
                s is in the right block)
                insert all children of remaining conjunction into right block
    } while (at least one insertion within loop)
}
```

Figure 5: Feasibility check of input and output constraints

```
procedure check_feasible (constraints) {
    I = generate_initial_encoding_dichotomies (constraints)
    D = remove_invalid_dichotomies (I, constraints)
    foreach (encoding-dichotomy d in D)
        raise_dichotomy (d, constraints)
    D = remove_invalid_dichotomies (D, constraints)
    foreach (encoding-dichotomy i ∈ I)
        if i is not covered by some d ∈ D
            return (INFEASIBLE)
    return (FEASIBLE)
}
```

Figure 6: Feasibility check of input and output constraints

6.3 Exact input and output encoding

The complete algorithm for satisfying both input and output constraints is shown in Figure 7. Following the generation of the initial encoding-dichotomies from the input constraints, those that violate output constraints are deleted. The remaining encoding-dichotomies are raised maximally. Any raised dichotomy that becomes invalid is deleted. If each of the initial encoding-dichotomies are covered by at least one of the valid and maximally raised dichotomies, we proceed to determine a minimum code length solution. All prime encoding-dichotomies are generated from the valid raised dichotomies. Using an exact unate covering algorithm, a minimum cover of the initial encoding-dichotomies by the valid prime encoding-dichotomies yields the exact solution.

An example is given in Figure 8. Notice that, given the initial encoding-dichotomies $(s_2; s_0s_1)$, $(s_0s_1; s_2)$, $(s_3; s_0s_1)$, $(s_0s_1; s_3)$, $(s_0; s_1)$, $(s_1; s_0)$, $(s_2; s_3)$ and $(s_3; s_2)$, the following are removed because they are invalid: $(s_0s_1; s_2)$ (it conflicts with $s_1 > s_2$), $(s_0s_1; s_3)$ (it conflicts with $s_0 = s_1 \vee s_3$) and $(s_0; s_1)$ (it conflicts with $s_0 > s_1$). By raising the remaining valid encoding-dichotomies one obtains the following raised encoding-dichotomies: $(s_1s_2; s_0s_3)$ (from the initial encoding-dichotomy $s_1; s_0$, since $s_1 > s_2$ forces s_2 into the left block and $s_0 = s_1 \vee s_3$ forces s_3 int the right block) that subsumes the valid encoding-dichotomy $(s_2; s_3)$, $(s_3; s_2s_1)$ (from $s_3; s_2$, since $s_1 > s_2$ forces s_1 into the right block), $(s_2; s_0s_1)$ and $(s_3; s_0s_1)$ (the last two are initial encoding-dichotomies unmodified by the raising process). Since each initial encoding-dichotomy is covered by some raised encoding-dichotomy, an encoding satisfying all constraints exists. It is found by computing the prime encoding-dichotomies obtained by the raised encoding-dichotomies and solving a unate covering problem to determine minimum code-length codes that satisfy the given constraints.

Theorem 6.2 *The algorithm shown in Figure 7 generates a minimum length encoding for a set of given input and output constraints, if a feasible solution exists.*

Proof Similar to that of Theorem 6.1. If a feasible solution exists it can be obtained from the maximally raised and valid encoding-dichotomies by generating prime encoding-dichotomies and finding a minimum covering of the initial encoding-dichotomies. ■

```
procedure exact_encode (constraints) {
    I = generate_initial_encoding_dichotomies (constraints)
    D = remove_invalid_dichotomies (I, constraints)
    foreach (encoding-dichotomy d ∈ D)
        raise_dichotomy (d, constraints)
    D = remove_invalid_dichotomies (D, constraints)
    foreach (encoding-dichotomy i ∈ I)
        if i is not covered by some d ∈ D
            return (INFEASIBLE)
    P = prime_dichotomy_generate (D)
    valid_primes = remove_invalid_dichotomies (P, constraints)
    mincov = minimum_cover (I, valid_primes)
    return (derive_codes (mincov))
}
```

Figure 7: Exact encoding constraint satisfaction

Face embedding constraints :
(s_0, s_1)
Dominance constraints :
$s_0 > s_1$ $\quad s_1 > s_2$
Disjunctive constraints :
$s_0 = s_1 \vee s_3$
Initial encoding – dichotomies :
$(s_2; s_0 s_1)$ $\quad (s_0 s_1; s_2)$ $\quad (s_3; s_0 s_1)$
$(s_0 s_1; s_3)$ $\quad (s_0; s_1)$ $\quad (s_1; s_0)$
$(s_2; s_3)$ $\quad (s_3; s_2)$
Raised encoding – dichotomies :
$(s_2; s_0 s_1)$ $\quad (s_3; s_0 s_1)$ $\quad (s_1 s_2; s_0 s_3)$
$(s_3; s_2 s_1)$
Prime encoding – dichotomies :
$(s_2; s_0 s_1 s_3)$ $\quad (s_2 s_3; s_0 s_1)$ $\quad (s_3; s_0 s_1 s_2)$
$(s_1 s_2; s_0 s_3)$
Minimum cover :
$(s_2 s_3; s_0 s_1)$ $\quad (s_1 s_2; s_0 s_3)$
Final encoding :
$s_0 = 11, s_1 = 10, s_2 = 00, s_3 = 01$

Figure 8: Example of exact encoding with input and output constraints

7 Bounded length encoding

Problem P-3 (*c.f.* Section 1) is defined as minimizing a cost function of the constraints that are satisfiable using a fixed number of bits. In practice, this problem is more relevant than problem P-2, where all constraints must be satisfied with the minimum number of bits. The reason is that encoding problems for logic synthesis often exhibit a trade-off between the code-length and the gain obtained by satisfying all constraints. Increasing the code-length may off-set what is gained by satisfying all constraints. For example, optimal encoding for finite state machines implemented by two-level logic may be viewed as the process of generating a set of mixed input and output constraints. Satisfying all the constraints may require a long code-length which translates into extra columns of the PLA than the minimum necessary. This often results in sub-optimal PLA area and also impacts the performance. The same reasoning applies to multi-level logic, where literal counts are used instead of cubes. Therefore, logic synthesis applications require an encoding algorithm that:

- Considers different cost functions.
- Minimizes a chosen cost function for encodings of fixed length.

There are three cost functions that are useful in such applications:

- The number of constraints satisfied.
- The number of product-terms in a sum-of-product representation of the encoded constraints.
- The number of literals in a sum-of-product representation of the encoded constraints [13].

We illustrate the meaning and technique of computation of the above-mentioned cost functions with an example. Consider the following input constraints: (e, f, c), (e, d, g), (a, b, d), (a, g, f, d). To satisfy all the constraints, a code-length of 4 bits is required. A solution is $a = 1010$, $b = 0010$, $c = 0011$, $d = 1110$, $e = 0111$, $f = 1011$, $g = 1100$. Suppose instead that the code-length is fixed to 3 bits. It must be the case that one or more input constraints are not satisfied, whatever 3-bit encoding is chosen. For a certain 3-bit encoding, the problem arises of computing its "goodness". For each input constraint I, define a logic function F_I whose on-set contains the codes of the symbols in the constraint and whose off-set contains the codes of the symbols not in the constraint. The unused codes are in the don't care set. For instance, given the previous encoding, the points in the on-set of $F_{(e,f,c)}$ are $(0111, 1011, 0011)$, those in the off-set are $(1010, 0010, 1110, 1100)$ while the don't care set contains the remaining unused nine codes. If a constraint is satisfied, two-level minimization of f_I yields a single product-term. If a constraint is not satisfied, there will be at least two product-terms in the minimized result. Thus, the number of product-terms is a measure of the satisfaction of the input constraints. For constraints arising from encoding problems of two-level logic, this is an appropriate cost function. In the procedure above, cost function evaluation requires a number of two-level logic minimizations. In practice this may be approximated by a single logic minimization of a multi-output Boolean function, where each constraint is represented by a unique output of a multiple output function. The number of literals of a two-level implementation of the constraints can be computed in the same way; here literals are counted instead of product-terms.

In Figure 9, a 3-bit encoding for the previous set of constraints is shown, together with the product-terms needed to implement the encoded constraints. The given 3-bit encoding violates 3 face constraints. 7 cubes and 14 literals are required to represent the encoded constraints.

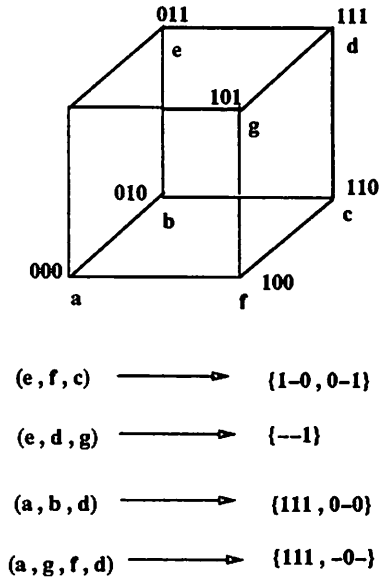

Figure 9: Example of cost function evaluation

7.1 Heuristic algorithm for input constraints

Consider the input constraint satisfaction problem where a code of length c bits is desired minimizing the number of violated constraints. This is an exact version of problem P-3. We require a selection of prime encoding-dichotomies that must have two properties. First, the primes must ensure that each symbol gets a unique code, that is, all the uniqueness constraints must be covered by the selected primes. Second, the fewest face constraints must be violated. The only apparent way this can be done is to enumerate all 2^{n-1} prime encoding-dichotomies (using n symbols) and then solve an exact weighted unate covering problem. Note that we desire a selection of primes that covers all uniqueness constraints, yet the cost function is related to the constraints specified. This approach is clearly infeasible on all but trivial instances of P-3.

Heuristic algorithms can be easily developed within the encoding framework presented in this paper. In the this subsection we describe a heuristic algorithm based on the concept of encoding-dichotomies to solve P-3 approximately.

The first phase of an exact solution to problem P-3 involves the enumeration of all 2^{n-1} encoding dichotomies that exist for n symbols. This step is termed *candidate encoding-dichotomy generation* (or *candidate generation* in short). The second phase is to determine a selection of a fixed number of these encoding dichotomies that minimize the desired cost function. This is termed *selection.* While *candidate generation* is clearly exponential in the input size, in the *selection* phase a polynomial (in the code-length c) number of sets of candidate encoding dichotomies have to be considered. A heuristic algorithm that avoids this enumeration of encoding-dichotomies while retaining the structure of the exact approach is detailed now.

The algorithm has three main phases: *splitting* of a set of symbols, *merging* of restricted encoding-dichotomies and selection of the c *best* restricted dichotomies for a subset of symbols. The splitting phase is used to divide the given encoding problem into two smaller problems, each using one less code bit. Assuming that each sub-problem is solved optimally, the solution for the original encoding problem

is generated by the steps of merging and selection.

Let a code of length c be desired for n symbols, $s_1, ..., s_n$. Consider a partition of the symbols into two groups $s_1, ..., s_k$ and $s_{k+1}...s_n$. Let D_1 be the $c-1$ best encoding-dichotomies restricted to $s_1, ..., s_k$. Similarly, let D_2 be the $c-1$ best encoding-dichotomies restricted to $s_{k+1}, ..., s_n$. Then, the candidate encoding-dichotomies for $s_1, ..., s_n$ is the set $D = \{(s_1...s_k; s_{k+1}...s_n)\} \cup D_1 \times D_2 \cup D_2 \times D_1$. The best selection of c dichotomies from D is used to obtain a desired encoding. By applying this technique recursively until each partition contains a single symbol, a bounded-length encoding is achieved.

Definition 7.1 *Let $S = s_1, s_2, ..., s_n$ be a set of symbols and let D be a set of encoding dichotomies using these symbols. Consider some subset of symbols, $P = s_{p_1}, s_{p_2}, ..., s_{p_k}$. The* **restricted dichotomies** *of D with respect to P are the elements of the set D_P of dichotomies obtained by removing all symbols not in P from each encoding-dichotomy $d \in D$.*

Splitting: We are interested in obtaining two sub-problems, each using one less code bit than the given problem does. In splitting the symbols into disjoint partitions, the fewest constraints should be violated. This is achieved by using a modification of the Kernighan-Lin [11] *partitioning* algorithm[6].

Each partition P can be considered as yielding a dichotomy, d_P. For example, the partition of n symbols into two blocks of symbols $\{s_1, ..., s_k\}$ and $\{s_{k+1}, ..., s_n\}$ gives $d = \{(s_1...s_k; s_{k+1}...s_n)\}$. Depending on the cost function being considered, each partition P is chosen to minimize the cost function evaluated using d_P. For example, if the number of face constraints is to be minimized, then P is chosen such that the fewest face constraints (restricted to the symbols being partitioned) are violated by d_P. If the number of literals (or cubes) is being minimized, then P is chosen such that the maximum number of restricted initial encoding-dichotomies are covered by d_P. This corresponds to minimizing the number of uncovered initial encoding-dichotomies. Thus, for the partitioning algorithm [11], the *nodes* are the symbols being partitioned and the *nets* are either face constraints or initial encoding dichotomies. Since the partitioning algorithm minimizes the number of nets that are cut, this suits the cost functions being considered here.

The procedure is executed recursively for the symbols in each of the parts. Each partition again yields candidate dichotomies restricted to the subset of symbols that appear in it. When only two symbols remain, a single dichotomy that corresponds to the uniqueness constraint between them is generated.

Consider the example shown in Figure 3, where an encoding of length 2 is required to minimize the number of literals in sum-of-product form. In the first step, at least four initial encoding-dichotomies must be violated by any partition for this example. Assume to choose the partition $P_1 = \{s_0, s_1, s_2, s_4\}$ and $P_2 = \{s_3\}$. Further partitioning the symbols in P_1 yields $P_{11} = \{s_0, s_4\}$ and $P_{12} = \{s_1, s_2\}$, which violates two of the initial encoding-dichotomies restricted to P_1 (numbered 1 and 4 in the example).

Merging: Here the restricted dichotomies generated from each of the sub-partitions, say P_1 and P_2, are merged to obtain a set of dichotomies that ensures unique codes for all the symbols in the merged partition, $P = P_1 \cup P_2$. Since the sets of symbols in P_1 and P_2 are disjoint, each of the dichotomies in P

[6]This step can also be performed by using the notion of incompatibility between encoding-dichotomies. The prime encoding-dichotomy that covers the maximum number of encoding-dichotomies is desired. Given the pairwise incompatibilities between encoding-dichotomies, this can be obtained by choosing the minimum cover of the pairwise incompatibilities (*cf.* Section 5.1). We do not employ this technique since the number of incompatibilities is often enormous. Additionally, the prime encoding-dichotomy is required to have a bounded number of symbols in each block, which requires a further modification to the approach.

is a union of one dichotomy each from P_1 and P_2. For that purpose, the cross-product of all dichotomies generated from each of the sub-partitions is formed.

Consider partitions $P_1 = \{s_0, s_1, s_2, s_4\}$ and $P_2 = \{s_3\}$ which are to be merged for the example of Figure 3. Assume the encoding dichotomies chosen (by recursive application of this algorithm) for P_1 are $D_1 = \{(s_0s_4; s_1s_2), (s_0s_2; s_1s_4)\}$. The only choice for P_2 is $D_2 = \{(s_3;)\}$. The merged dichotomies to be considered are $D = \{(s_0s_1s_2s_4; s_3), (s_0s_3s_4; s_1s_2), (s_0s_4; s_1s_2s_3), (s_0s_2s_3; s_1s_4), (s_0s_2; s_1s_3s_4)\}$. The best encoding of length 3 is chosen from this set by the next step.

Selection of best restricted dichotomies: The objective here is to generate those combinations of dichotomies from each partition that maximally recover the constraints violated by the partitions in the first step, while covering all the uniqueness constraints. It is important to note that when the best selection of encoding-dichotomies restricted to a subset of symbols is sought, a global view of constraints (and cost function) must be employed. This is done as follows for a subset of symbols $P = \{p_1, ...p_k\}$ with candidate restricted encoding-dichotomies D_p. A cover of size c_{D_p} is desired. The constraints of the entire problem are first restricted to the symbols $p_1, ...p_k$. Each selection of c_{D_p} encoding-dichotomies from D_p is evaluated using the approach mentioned in the previous section. The cover that minimizes the given cost function is chosen as the c_{D_p} best restricted encoding-dichotomies. This selection ensures minimization of the cost function for all the symbols after the merging step is completed.

Continuing with the example of Figure 3, assume the merging step discussed above. The 3 best encoding-dichotomies selected are $(s_0s_1s_2s_4; s_3)$, $(s_0s_2; s_1s_3s_4)$ and $(s_0s_4; s_1s_2s_3)$. This is done by evaluating all selections of size 3 from the set D that cover all uniqueness constraints and minimize the literal count. In the general case the number of evaluations can be restricted to some fixed number to reduce the search space.

While some selection of partitions, in the splitting step, and some selection of dichotomies, in the selection step, yield the best solution, it may not necessarily be obtained without a complete evaluation of all the possibilities in the partitioning and selection phases. However, a straightforward implementation of the algorithm, without a complete branch and bound search, has shown promising results.

8 Extensions to other encoding constraints

In this section we illustrate that the formulation presented in Section 6 provides a uniform framework for the satisfaction of various other known encoding problems.

8.1 Input encoding don't cares

The notion of an encoding don't care was first described in [6], where it was pointed out that for a multi-valued literal, any binary coded implementation of the literal which contains the *reduced implicant* and is contained by the *expanded implicant* in which it occurs, is valid. An example of how encoding don't cares are generated in the two-level case is given in [14]. A face constraint containing symbols a, b and e and with symbols c and d as encoding don't cares is denoted $(a, b, [c, d], e)$. This constraint specifies that symbols a, b, e must be assigned to one *face* of a binary n-dimensional cube, with the don't care symbols c, d free to share or not the same face. and no other symbol sharing the same face. These encoding don't cares have been shown to be essential for determining good factors in deriving a multi-level implementation of a given multi-valued description [13].

A simple example shows that suboptimal solutions of P-2 are computed when input encoding don't cares are disregarded. Given the set of face constraints (a,b), (a,c), (a,d), $(a,b,[c,d],e)$, a minimum cover of primes contains 3 primes, e.g. $(a,b,e;d,f)$, $(a,c,d;b,e,f)$, $(a,b,d;c,e,f)$. If instead the encoding don't cares are forced to be in the face constraint, i.e. $(a,b,[c,d],e)$ is replaced by (a,b,c,d,e) then a minimum cover of primes contains 4 primes, e.g. $(a,b,c,d,e;f)$, $(a,b,c;d,e,f)$, $(a,c,d;b,e,f)$, $(a,b,d;c,e,f)$. Also in the case that the encoding don't cares are forced not to be in the face constraint, i.e. $(a,b,[c,d],e)$ is replaced by (a,b,e) a minimum cover of primes contains 4 primes, e.g. $(a,b,e;c,d,f)$, $(a,b,c;d,e,f)$, $(a,d;b,c,e,f)$, $(a,c,d;b,e,f)$.

The framework described in Section 6 naturally handles encoding don't cares in the face constraints. Consider the face constraint $(s_0s_1s_3[s_5])$, which implies that s_5 may or may not be chosen to be on the same face as s_0, s_1 and s_3 in the final encoding. Converting this constraint to initial encoding-dichotomies is simply a matter of not generating the encoding-dichotomies $(s_0s_1s_3;s_5)$ and $(s_5;s_0s_1s_3)$. The absence of these dichotomies enables s_5 to be either inside or outside the face that includes s_0, s_1 and s_3 depending on the minimality of the encoding. In presence of encoding don't cares, a prime may give a bi-partition of a proper subset of the symbols. For instance, if we consider the set of face constraints of the previous example $(a,b,e;d,f)$, $(a,c,d;b,e,f)$, $(a,b,d;c,e,f)$, the prime encoding dichotomies generated by the extended definition of compatibility are: $(a,b,e;f)$, $(a,b,e;d,f)$, $(a,b,e;c,f)$, $(a,b;c,d,e,f)$, $(a,c;b,d,e,f)$, $(a,d;b,c,e,f)$, $(a,b,c;d,e,f)$, $(a,c,d;b,e,f)$, $(a,b,d;c,e,f)$, $(a,b,c,d;e,f)$. A minimum cover of 3 primes can be extracted out of them, as shown before.

The algorithms described for the feasibility check and exact encoding, shown in Figures 5, 6 and 7 respectively, extend naturally to encoding don't cares. Note that the satisfiability check algorithm described in [9] cannot be easily extended to handle encoding don't cares without a significant penalty in run-time. The encoding algorithm presented in [24] also cannot be extended to handle don't cares.

8.2 Distance-2 constraints

In [22, 21, 8] a condition for easy and full sequential testability involved satisfying encoding constraints which specified that the encoding of a pair of states must be at least distance-2 apart. This condition may be easily satisfied by ensuring that at least two prime encoding-dichotomies are selected in the minimum cover that each have the two states in different blocks. Suppose that we want to keep a distance of 2 between the codes assigned to states a and b. Suppose that, of all primes, the pairs p_1, p_2 and p_3, p_4 have the two states in different blocks. One at least of the two pairs must be chosen in a final cover. This can be enforced by adding to the binate covering formulation the clauses

$$(p_1 + \overline{b}_1)(p_2 + \overline{b}_1)(p_3 + \overline{b}_2)(p_3 + \overline{b}_2)(b_1 + b_2),$$

where b_1 and b_2 are two new additional literals (columns of the table) that do not represent encoding columns, but enforce the choice of other columns in a solution.

8.3 Other constraints

In [18], conditions for improved sequential testability were given which require the encodings to ensure that certain face embedding constraints must not be satisfied in addition to the face embedding constraints that must be satisfied. They are called non-face constraints. These constraints specify that a set of symbols is to be assigned to one *face* of a binary n-dimensional cube, with at least another symbol not in

the set sharing the same face. A non-face constraint involving symbols a, b and e is denoted by $)a, b, e($. An encoding satisfying the face constraints (a, b), (b, c, d), (a, e) and (d, f) and the non-face constraint $)a, b, e($ is given by $a = 011, b = 001, c = 101, d = 100, e = 111, f = 110$. The face spanned by the codes of the symbols in the non-face constraint is $--1$ and contains also the code of the symbol c.

These constraints can be captured in our framework by extending the final covering step to a binate covering, where sets of prime encoding-dichotomies that cover the encoding-dichotomies yielded by the face embedding constraints which should not be satisfied are not allowed.

For each non-face constraint all minimal sets of prime dichotomies that cover the related face constraint are computed. In the case of $)a, b, e($ the related face constraint is (a, b, e) and the encoding dichotomies are $d_1 = (a, b, e; c), d_2 = (a, b, e; d), d_3 = (a, b, e; f)$. Suppose that the minimal sets of prime dichotomies that cover d_1, d_2, d_3 are $\{p_1\}, \{p_3, p_4\}, \{p_3, p_5, p_6\}$. We must add to the unate clauses of the covering expression, these additional negative clauses:

$$\overline{p}_1(\overline{p}_3 + \overline{p}_4)(\overline{p}_3 + \overline{p}_5 + \overline{p}_6).$$

For instance, $(\overline{p}_3 + \overline{p}_5 + \overline{p}_6)$ says that, when selecting for a minimum cover two primes in p_3, p_5, p_6, the third one cannot be chosen.

8.4 Limitations of dichotomy-based techniques

This section has illustrated how new classes of encoding constraints, together with face and output constraints, can be accomodated in the dichotomy-based frame. It is legitimate to ask what kind of constraints, if any, cannot be naturally phrased with the language of dichotomies.

Such an example of unwieldy encoding constraints are *chain* constraints. They can be found in [1], where a technique for implementing finite state machines using counter-based PLA structures is presented. State assignment is reduced there to a step of deriving face and chain constraints and a step of satisfying them. A chain constraint requires that increasing binary numbers be assigned to the codes of the ordered sequence of states. The first element in the chain can be given any code. For instance, a chain constraint involving the ordered sequence $a, b, c, d, e, f, g, h, i$ is denoted by $(a-b-c-d-e-f-g-h-i)$ and is satisfied by the encoding $a = 0010$, $b = 0011$, $c = 0100$, $d = 0101$, $e = 0110$, $f = 0111$, $g = 1000$, $h = 1001$, $i = 1010$. For every pair of adjacent states in the chain the code of the right state is equal to the code of the left state increased by one in binary arithmetic. As an example of encoding problem with face and chain constraints, consider the face constraints (b, c), (a, b), and the chain $(d - b - c - a)$. A satisfying assignment is: $a = 00, b = 10, c = 11, d = 01$.

Even though it is possible, for a given code length, to add to the covering expression clauses that impose the chaining conditions, a straighforward solution seems to require a computationally expensive implicit enumeration. The question of whether tighly coupled constraints, such as chain constraints, can be couched usefully in the frame provided here is left open.

9 Results

Table 1 gives the results of using the exact encoding algorithm on a set of examples using both input and output encoding constraints. These constraints were generated using an extension of the procedure described in [6] that also generates good disjunctive effects. The number of valid prime encoding-dichotomies is shown in the third column. As seen from the table, all the examples with less than 50000

Name	# States	# Primes	# Bits	Time (secs)
bbsse	16	1449	7	20
cse	16	201	7	3
dk16	27	24316	12	1050
dk16x	27	6205	12	530
dk512	15	35	9	1
donfile	24	673	12	17
ex1inp	20	2023	9	45
keyb	19	189	9	4
kirkman	16	54	11	8
master	15	972	5	4
planet	48	> 50000	*	*
s1	20	469	7	10
s1a	20	50	7	3
sand	32	2481	11	88
tbk	32	13	12	41
vmecont	32	> 50000	*	*

* indicates results not available
Table 1: Exact input and output encoding

primes completed in very little CPU time on a DEC 3100 work-station. In the cases of *planet* there were only nine dominance constraints and no disjunctive constraints, which led to almost no decrease in the number of primes generated from the face constraints (exponential in the worst case). In the case of *vmecont* there were only eight different face constraints (six of them had only two states), which led to a huge number of primes being generated from the large number of unimplied uniqueness constraints. The previous approach suggested for prime generation in [25] could not complete on any of the examples.

Table 2 compares an implementation of the heuristic algorithm described in Section 7.1 with NOVA [24] for two-level implementations. The input constraints have been generated calling the two-level multiple-valued logic minimizer ESPRESSO-MV [17]. In this case, the number of face constraints satisfied using the minimum possible length for encoding are compared. While both programs perform comparably with regard to the number of constraints satisfied, our approach has a significant advantage compared to NOVA whith respect to the number of cubes needed to implement in two-level form the input constraints. This cost function is very important because it measures the advantage of satisfying a subset of input constraints in a fixed code-length more precisely. Our algorithm in almost all cases needs fewer cubes than NOVA. On the benchmark set it requires on average 13% fewer cubes and in some cases the gain is more than 20%.

Table 3 compares our approach to simulated annealing for multi-level examples. Input constraints with don't cares are generated by the multiple valued multi-level synthesis program MIS-MV [13] with the number of factored form literals in the encoded implementation as cost function (in practice, the number of literals in a sum-of-product representation of the encoded constraints is used as an approximation to this cost function). Because of the presence of encoding don't cares and the cost function of

Name	States	# Constraints	Constraints		Cubes	
			NOVA	ENC	NOVA	ENC
bbsse	16	5	3	3	12	8
cse	16	12	8	8	24	18
dk16	27	33	25	20	43	48
dk512	15	10	8	9	12	11
donfile	24	24	8	11	48	39
ex1	20	11	8	8	19	19
kirkman	16	25	9	9	58	58
master	15	3	3	3	3	3
planet	48	12	12	12	12	12
s1	20	14	14	14	14	14
sand	31	7	6	6	8	8
styr	30	18	14	14	29	26
tbk	32	98	44	39	284	237
viterbi	68	6	6	6	6	6
vmecont	32	40	24	25	81	67

Constraints: Number of constraints to be satisfied
Constraints: Number of satisfied constraints
Cubes: Number of cubes in a two-level implementation of the constraints
NOVA: Encoding using NOVA [24], minimum code length
ENC: Heuristic encoding, minimum code length

Table 2 : Two-level heuristic minimum code length input encoding

Name	States	Literals		Time	
		SA	ENC	SA	ENC
bbsse	16	162	164	3017	175
cse	16	229	236	3969	234
dk16	27	336	380	27823	1523
dk512	15	82	85	2090	138
donfile	24	154	172	16265	935
kirkman	16	201	229	2621	322
master	15	392	398	2069	423
s1	20	280	304	16297	833
†sand	31	763	737	1926	2332
†tbk	32	560	498	3774	4090
†viterbi	68	327	322	860	1013
†vmecont	32	378	364	2074	2883

SA: Simulated annealing (10 moves per step)
ENC: Heuristic encoding in minimum code length
Time SA / END: Time ratio for SA vs. ENC; includes run time for minimization script [13]
†: SA cannot complete with 10 moves per step; SA limited to 4 steps per move

Table 3 : Multi-level heuristic minimum code length input encoding

literals, simulated annealing was the only known algorithm for solving this problem. We use two sets of experiments to compare the effectiveness of our heuristic bounded-length algorithm versus the version of simulated annealing algorithm implemented in MIS-MV. Minimum-length encoding is always used. MIS-MV is run using a script that invokes the constraints satisfaction routine six times; five times to perform a cost evaluation that drives the multi-valued multi-level optimization steps and one final time to produce the actual codes that replace the symbolic inputs [13]. Simulated annealing is called the first five times with 1 pairwise code swap per temperature point, while the last call performed 10 pairwise code swaps per temperature point. Simulated annealing does not complete on the larger examples with 10 pairwise swaps per step. These examples are marked with a † in the table, and only 4 swaps were allowed per temperature step for these examples. When using our heuristic algorithm, the full-fledged encoder is called all six times. See [13] for a detailed explanation of the scripts.

As can be seen from Table 3, our algorithm on average performs a little better than simulated annealing in terms of literal counts. This is significant especially in the large examples, where it reduces the literals counts up to 10% further than simulated annealing. When our algorithm does worse it is within 5% of the simulated annealing result. However, a significant parameter here is the amount of time taken. Simulated annealing consumes at least an order of magnitude of time (two orders or more for larger sized examples) more than our algorithm when a better quality solution is desired, i.e. using 10 swaps per step. On attempting to reduce the runtime to be comparable to our approach, a noticeable loss of optimization quality compared to our approach may be observed in the table. Further improvements to the heuristic encoding algorithm are still under investigation.

10 Conclusions

This paper has presented a comprehensive solution to the known constraint satisfaction problems arising in the two-step encoding paradigm defined in the introduction. A theory has been provided and a set of applications fully developed. We have shown that the problem of determining a minimum length encoding to satisfy both input and output constraints is NP-complete. Based on an earlier method for satisfying input constraints [25], we have provided the first formulation of an algorithm that determines the minimum length encoding that satisfies both input and output constraints. It is shown how this algorithm can be used to determine the feasibility of a set of input and output constraints in polynomial time in the size of the input. This formulation also provides a uniform framework for solving a variety of other known encoding constraints. While all previous exact formulations for the input encoding problem have failed to provide efficient algorithms, an algorithm that efficiently solves the input and output encoding constraints exactly for a large standard benchmark set has been described. An effective heuristic procedure, which uses the framework provided, for solving input encoding constraints in both two-level and multi-level implementations is also demonstrated.

References

[1] R. Amann and U. Baitinger. Optimal state chains and state codes in finite state machines. *IEEE Transactions on Computer-Aided Design*, February 1989.

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

[3] R. Brayton, A. Sangiovanni-Vincentelli, and G. Hachtel. Multi-level logic synthesis. *The Proceedings of the IEEE*, february 1990.

[4] K-T. Cheng and V.D. Agrawal. State assignment for initializable synthesis. In *The Proceedings of the International Conference on Computer-Aided Design*, November 1989.

[5] G. Cybenko, D. Krumme, and K. Venkataraman. Fixed hypercube embedding. *Information Processing Letters*, April 1987.

[6] G. DeMicheli. Symbolic design of combinational and sequential logic circuits implemented by two-level logic macros. *IEEE Transactions on Computer-Aided Design*, October 1986.

[7] G. DeMicheli, R. Brayton, and A. Sangiovanni-Vincentelli. Optimal state assignment for finite state machines. *IEEE Transactions on Computer-Aided Design*, July 1985.

[8] S. Devadas, H-T. Ma, R. Newton, and A. Sangiovanni-Vincentelli. Synthesis and optimization procedures for fully and easily testable sequential machines. In *The Proceedings of the International Conference on Computer-Aided Design*, November 1987.

[9] S. Devadas and R. Newton. Exact algorithms for output encoding, state assignment and four-level Boolean minimization. *IEEE Transactions on Computer-Aided Design*, January 1991.

[10] C. Duff. Codage d'automates et theorie des cubes intersectants. *Thèse, Institut National Polytechnique de Grenoble*, March 1991.

[11] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, February 1970.

[12] D. Krumme, K. Venkataraman, and G. Cybenko. Hypercube embedding is NP-complete. In *Proceedings of SIAM Hypercube Conference*, September 1985.

[13] L. Lavagno, S. Malik, R. Brayton, and A. Sangiovanni-Vincentelli. MIS-MV: Optimization of multi-level logic with multiple valued inputs. In *The Proceedings of the International Conference on Computer-Aided Design*, November 1990.

[14] L. Lavagno, T. Villa, and A. Sangiovanni-Vincentelli. Advances in encoding for logic synthesis. In *Progress in Computer Aided VLSI design, G. Zobrist ed.* Ablex, Norwood, 1991.

[15] M. Marcus. Derivation of maximal compatibles using Boolean algebra. *IBM Journal of Research and Development*, November 1964.

[16] R. Rudell. Logic synthesis for VLSI design. *Tech. Report No. UCB/ERL M89/49*, April 1989.

[17] R. Rudell and A. Sangiovanni-Vincentelli. Multiple-valued minimization for PLA optimization. *IEEE Transactions on Computer-Aided Design*, September 1987.

[18] A. Saldanha and S. Chandra. Synthesis for improved sequential controllability. *Unpublished manuscript, U.C. Berkeley*, November 1990.

[19] A. Saldanha, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli. A uniform framework for satisfying input and output encoding constraints. June 1991.

[20] G. Saucier, C. Duff, and F. Poirot. State assignment using a new embedding method based on an intersecting cube theory. In *The Proceedings of the Design Automation Conference*, 1989.

[21] P. Srimani. MOS networks and fault-tolerant sequential machines. *Computers and Electrical Engineering*, 8(4), 1981.

[22] P. Srimani and B. Sinha. Fail-safe realisation of sequential machines with a new two-level MOS module. *Computers and Electrical Engineering*, 7, 1980.

[23] J. Tracey. Internal state assignment for asynchronous sequential machines. *IRE Transactions on Electronic Computers*, August 1966.

[24] T. Villa and A. Sangiovanni-Vincentelli. NOVA: State assignment for optimal two-level logic implementations. In *IEEE Transactions on Computer-Aided Design*, September 1990.

[25] S. Yang and M. Ciesielski. Optimum and suboptimum algorithms for input encoding and its relationship to logic minimization. *IEEE Transactions on Computer-Aided Design*, January 1991.