Boolean Constrained Encoding: a new formulation and a case study

Ney Laert Vilas Calazans
Instituto de Informática – PUCRS – Porto Alegre, RS – BRAZIL
e-mail: calazans@brpuvsr.bitnet

Abstract

This paper provides a new, generalized approach to the problem of encoding information as vectors of binary digits. We furnish a formal definition for the Boolean Constrained Encoding problem, and show that this definition encompasses many particular encoding problems found in VLSI design, at various abstraction levels involving compatibility classes and/or equivalence classes in the original symbol set encoded, by allowing symbols to be cubes of a Boolean space, instead of the usual minterms. Besides, we state a unifying optimization framework to represent required constraints which is more general than previous efforts. The framework is based upon a new definition of the pseudo-dichotomy concept, and is adequate to guide the solution of encoding problems through the satisfaction of constraints extracted from the original problem statement. An encoding problem case study is presented, the state assignment of synchronous finite state machines with the simultaneous minimization of state and output functions. The problem is solved by a comparison with well-established approaches to solve this problem in two separate steps, shows that our solution is competitive with other published results. However, the case study is intended to show the feasibility of the Boolean constrained encoding problem formulation.

1 Introduction

Encoding is a fundamental step of numerous problems in computer science, computer design and VLSI design problems. The optimal solution of any such problem depends on the satisfaction of a set of constraints as well as on an objective optimization criterion, all of which must be defined in terms of the original problem statement. Encoding is basically a translation process, where a set of symbols is mapped into Boolean vectors. Many of the general approaches to encoding in VLSI design appeared as a by-product of solutions to the state assignment problem for finite state machines (FSMs). Most solutions to this problem assume encodings that are injective functions (one-to-one mappings) from the state set into a set of Boolean vectors of a given fixed length [9, 10]. Although the use of injective functions is useful for the state assignment problem alone [4], it represents a severe limitation if more powerful encoding strategies are intended to be used for other purposes. For example, suppose that the set of symbols to be encoded has a structure that allows the identification of equivalence classes in it. In order to capture such a structure, we must allow encodings that are not injective, so that every symbol in an equivalence class can be mapped into a unique Boolean vector. A more complicated case arises when the set of symbols contains compatibility classes. Here, the encodings must be allowed to be both non-injective and non-functional, so that the intersection of overlapping classes is related to more than one Boolean vector. Since equivalence classes and compatibility classes are so commonly found in the structure of VLSI design problems, it is useful to consider them in the scope of encoding problems.

In this paper, we propose a general approach to constrained encoding problems. In Section 2, we introduce the needed basic definitions and the Boolean Constrained Encoding (BCE) problem, a formalization for which encompasses previously proposed formalisms [9, 10], and which additionally allows that compatibility classes present in the symbol set be captured in the encoding process. Aim ing at the construction of more powerful resolution methods for encoding problems, we propose a unified framework for representing encoding constraints in Section 3. This framework is based on a new statement of the well-known concept of (pseudo-)dichotomies. Several publications have reported the use of dichotomies to model the state assignment problem in both asynchronous and synchronous [11, 3] FSMs, with their use to cope with other encoding problems such as two-way network partitioning and two-layer via minimization [10]. Our definition stresses the relationship between the concept and the underlying algebra of switching functions, and is also more comprehensive than that in previous approaches. The main goal of the framework is to provide a unique representation for constraints found in a comprehensive class of encoding problems, and which are related to several aspects of VLSI design, such as area and delay optimization and testability enhancement. The solution of practical problems as instances of the BCE problem depends on how easily the former can be mapped to the latter. Section 4 presents a discussion on how to express classes of constraints commonly found in encoding problems using the pseudo-dichotomy framework. Other constraint classes are less often in the scope of these problems are analyzed as well, since they will be useful in the search for encodings intended to represent compatibility classes arising in the symbol set. The utility of the mapping process is that the framework represents the original problem in a standard form, amenable to treatment by common constraint satisfaction algorithms. Section 5 illustrates the BCE problem resolution process based on the unified framework through a case study, the state assignment of synchronous FSMs with the simultaneous consideration of state and output minimization. Section 6 also introduces a computer program implementation for solving the case study problem, and presents benchmark results. Finally, Section 7 lists a set of conclusions, and points directions for future work.

2 Basic Definitions and the BCE Problem

Definition 2.1 (Partition) A set of two sets $S$ and $T$, a binary relation $(S,T,r)$, is a partition of $T$ if $S$ is finite and one-to-one. The image $r(S)$ of an element $s$ of $S$ is a block of partition $r$.

Let $S$ be a set specified as a Cartesian product $S = \times_{i=1}^{n} S_i$ and $L$ be a set of integers between 0 and $r - 1$, with an associated lattice structure $(L,\leq,\wedge,0,\wedge,0,\wedge,0)$ under the operations $\vee$ and $\wedge$, with least element 0 and greatest element $r - 1$. Assume also that $S = (0,1)$.

Definition 2.2 (Lattice exponentiation function) Let $X$ denote the vector $(x_1, \ldots, x_n)$ of variables taking values on $S$; given a variable $x_i \in X$ and a subset $C_i \subseteq S_i$, we define the lattice exponentiation function as the discrete complete function $c_i^*(C_i) = \{(r - 1) i : x_i \in C_i, 0 \text{ otherwise.}\}$

Definitions 2.3 (Cube function) A cube function or simply a cube is a discrete complete function $c : S \rightarrow L$, where the values $c(x)$ are computed by the expression $c(x) = \wedge_{i=0}^{m} c_i^*(C_i), \text{ with } L = L, C_i \in S_i$.

The lattice element $l$ is called the weight of the cube. If $c(x)$ is such for all $C_i \subseteq S_i$, then $c(x)$ is a minterm. Given two cubes, $x_1(x) = \wedge_{i=0}^{m} c_i^*(C_i), \text{ with } l, C_i \in S_i, \text{ and } d(x) = \wedge_{i=0}^{m} d_i^*(D_i), \text{ with } l, D_i \in S_i$, their supercube is a cube defined as $p(x) = (l \wedge m) \wedge_{i=0}^{m} (c_i^*(C_i) \cup D_i)$.

The supercube definition is immediately extendable to sets of cubes with cardinality bigger than two. The cubes $c(x)$ and $d(x)$ are disjoint if there is no $C_i \subseteq S_i$, for which $c_i(x) = 1$ and $d_i(x) = m$, or if $l = 0$ or $m = 0$. The size of a cube $c(x)$ is $|X| |c(x)| = 1$, if $l = 0$, otherwise the size is 0. The satisfying set of $c$ is the set of Boolean vectors $X \subseteq c(x) = 1$, if $l \neq 0$, otherwise $X$ is the empty set. Every element in this set is said to satisfy the cube function $c$. A switching cube function $c$ whose cubed domain is $S = B^n$ and whose codomain is $L = B$, for some integer $n$.

The usual definition of a cube as a product of literals is a limited interpretation of the formal concept of satisfying set of a cube [2]. The most important of the definitions is that of encoding.

Definitions 2.4 (Encoding or Assignment) Given a positive integer $n$, an assignment or encoding of a set $S$ is a mapping or complete function $c : S \rightarrow P(B^n)$, where $P(B^n)$ stands for the power set of the set of all subsets of $B^n$. The integer $n$ is called the length of the assignment. For every $c \in S$, the image $c(x)$ is called the code of $x$. Two codes $e(a), e(t)$ are disjoint if $e(a) \cap e(t) = \emptyset$, otherwise the codes are intersecting. An assignment that is an injection is where codes are pairwise disjoint in an injective encoding, in which case $V(x \in S), x \neq t \implies e(a) \cap e(t) = \emptyset$. Any assignment that is not injective is called non-injective.
A cube assignment or cube encoding of $S$ is an assignment of a set $S$ where every cube $c(s)$ is an encoding of some cube, i.e., there is a switching cube $c(X)$ over $B^n$ such that $c(x) = 1 \Leftrightarrow X \in S$. Let the codes of a cube assignment $c$ be represented as three-valued vectors $v = v_0, \ldots, v_N$ and the cardinality of $S$ be $S$. The three-valued column vectors are obtained by assigning all elements $v_i$ for some $i$, over all codes $c(a), a \in S$, have the form $(v_0, \ldots, v_{i-1}, 0, v_{i+1}, \ldots, v_N)^T$ and are called columns of the encoding. The exponent $I$ notation indicates the transpose of the row vector, expressing the usual vertical interpretation of the term column.

A functional assignment or functional encoding of $S$ is an assignment where every code is a singleton. A functional assignment is redefined as a function $f : S \rightarrow B^n$, without loss of generality. We call assignments that are not functional non-functional.

The concept of assignment is limited here to fixed-length codes. We may thus refer to the encoding length also as the code length. In the rest of this paper, we restrict attention to cubes or to the more restrictive functional encodings. Correspondingly, Boolean codes will be represented as either Boolean tuples or as three-valued vectors. We adopt herein the simplified notation $v = \ldots, v_m$ to represent a tuple of the set $B^n$, and call it a Boolean vector.

Definition 2.5 (Discrete function satisfaction) Let $f : S \rightarrow L$ be a discrete function and $I : B^n \rightarrow B^n$ be a general switching function. We say that $f$ satisfies $I$ under the two assignments $\psi : S \rightarrow B^n$ and $\psi : L \rightarrow B^n$ if these assignments are such that $\forall s \in S \exists s \in S \exists s \in S f(s) = I I(\psi(s)) = \psi(l)$. We may now state the BCE problem. Our approach is to associate the columns of an encoding to constraints, which imply the well-known and efficient column-based encoding strategy [12].

Problem Statement 2.1 (Boolean Constrained Encoding) Consider the sets $s = \{s_0, \ldots, s_{N-1}\}$ and $C = \{c_0, \ldots, c_{M-1}\}$, where the elements $e_c \in C$ are switching functions $f_1 : B^n \rightarrow B$. Associate with each $f_1$ a positive row number $c_1$ and a discrete function $g_1$, such that $g_1 = \{g_1(s) = 1 \Leftrightarrow f_1(s) = 0\}$. We call the elements of $S$ symbols, and the elements of $C$ encoding constraints on the symbols of $S$. The number $c_1(s)$ is the gain of $f_1$, while $g_1$ is the encoding constraint satisfaction function of $f_1$. A constraint $f_1$ is satisfied by a set $E \subseteq \{B^n\}$ if there is an element $e_c \in E$ such that $g_1(e) = 1$. The satisfaction of a constraint $f_1$ by $E$ is indicated here, with a little notation abuse by $g_1(E) = 1$. The Boolean constrained encoding problem consists in finding a function $h : S \rightarrow P(B^n)$, where $P(B^n)$ is the power set of $B^n$, and such that $h$ is minimized, and the gain $c(h)$ is maximized, where $c(h)$ is defined as

$$c(h) = \sum_{i=0}^{M-1} c_1(s_i) \cdot g_1(s_i)$$

Function $h$ is denoted an encoding function or simply an encoding, while the $k$ is called the encoding length.

Example 2.1 (Input assignment) Consider the approach proposed in [4], that solves the state assignment problem by approximating it as an input assignment problem, i.e. by respecting the face embedding constraints generated by symbolic minimisation. Consider, for instance, the FSM $A = (F, Q, O, X, \lambda)$ where $F = I \subseteq I \subseteq B^n$, $T = (a, b, c, d)$, and where $\lambda$ (the transition function) and $\lambda$ (the output function) are given by the flow table in Figure 2.1, where also appears the grouping of entries obtained by symbolic minimisation.

<table>
<thead>
<tr>
<th>$0$</th>
<th>$1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td>$b$</td>
<td>$c$</td>
</tr>
<tr>
<td>$c$</td>
<td>$d$</td>
</tr>
</tbody>
</table>

Figure 1: Flow table and input constraints for FSM $A$.

To each grouping corresponds a face embedding constraint. These are associated to the present state of each entry group in the Figure 2.4, and are not just constraints, but only four distinct ones, which are $(a, b), (c, d), ((a, b), (c, d)), ((b, c), (d, e)), ((e, f), (d, e))$. Remember the interpretation of these constraints: two symbol encoding constraints in the set $E$ with one row encoding, and $k^2$-level implementation [12]. Then, satisfying a constraint guarantees that all groupings associated with this constraint can be performed in the final implementation. If all constraints are finally satisfied, we had better choose to satisfy constraints that are repeated many times, since this will hopefully lead to a greater percentage of groupings from all those predicted by symbolic minimisation.

The constraint satisfaction function $g_1$, in this example, has an expression which is identical to the corresponding function $f_1$, but defined over a larger domain. This is a consequence of the fact that face embedding constraints are satisfied by a single encoding column of the result. However, this does not account for the general case. In some problems, a constraint is satisfied if a face embedding associated with $f_1$ appears in the symbolically minimised cube table. This is justified by the fact that each entry set in the symbolically minimised cube table is associated with one row encoding.

Let us now give general interpretations for the elements in the BCE problem statement. Consider the set $S$ of cube assignment $c$ such that $c$ is a cube assignment, and $c$ is a cube assignment that is compatible with the constraints in $C$. The encoding constraints $c_1$, on the other hand, map a Boolean vector with the same cardinality as the set of symbols to a binary digit. The most frequent interpretation for this function is that it tells whether or not a bit column participates in the satisfaction of the encoding constraint. To each $f_1$ the problem statement associates $g_1$, a function that characterises the encoding constraint. It is through $g_1$ that the behavior of the distinct encoding constraint classes can be accounted for. The encoding constraint satisfaction function $g_1$ tells if the encoding constraint $f_1$ is satisfied or not by every possible functional encoding of $k$ bits.

The encoding function $h$ is the solution of the problem. It associates a set of binary $k$-tuples with each symbol in $S$, unlike previous propositions [9, 10], which associated a single $k$-tuple with each symbol. This is a major generalisation of the statement, that allows non-injective and/or non-functional encodings to be obtained.

The multiplier inside the summation in the expression for $c(h)$, i.e. the expression $g_1 \left( \sum_{i=0}^{M-1} (h(s_i)) \right)$, evaluates to 1 iff the constraint $f_1$ is satisfied by the final encoding. Otherwise, it evaluates to 0.

The encoding function $h$ is a NP-hard problem, i.e. a polynomial-time function cannot be found for an encoding of $n$ variables such that, for every encoding, it minimises $c(h)$. Thus a restricted version is called Complete (Boolean) Constrained Encoding (CBCE). Another restricted version of the Boolean constrained encoding problem is obtained as follows: establish values for $k$ (often the minimum possible), looking then for an encoding with length $k$ that minimises $c(h)$. This problem is called Partial (Boolean) Constrained Encoding (PBCE).

3 A Unified Constraint Framework

Encoding constraints were modeled in Problem Statement 2.1 as switching functions of $n$ variables. This choice is general enough to represent the two most kinds of constraints that may arise in several levels of abstraction in VLSI designs. The widely accepted definitions of (pseudo-)dichotomies is not as general as the definition of encoding constraints, and does not provide a more general definition of the former. On the other hand, the function $g_1$ accounts for the satisfaction of the constraints $f_1$ across an
arbitrary set of columns of the encoding. This justifies the proposal of an ad hoc framework that allows us to concentrate on one of more than one constraint of the final encoding.

3.1 The Pseudo-Dichotomy Concept
Pseudo-dichotomies had originally little or no algebraic structure associated to them. No addition or product of dichotomies can be defined, although there is a way of combining and splitting concepts like compatibility and covering between dichotomies have found their way in previous publications [1, 3]. Our definition links the concept to well-defined algebraic structures, allowing a more formal and thorough treatment of its applications. A pseudo-dichotomy is a concept to model single constraints in boolean encoding problems. In general, these constraints consist on indications to make the code of symbols disjoint or intersecting. Two-block partitions are adequate to model such a behavior. A two-block partition is intended to make bits on one block distinct from the bits in the other block, thus making the code of symbols in distinct blocks disjoint. However, the partition definition implies that all elements of a block may be contained in one block. This can be relaxed by taking partitions of subsets of the symbol set. On the other hand, the rules to use such partition-like structures may also vary from one problem to another.

We propose pseudo-dichotomies as algebraic structures composed by a two-block partition-like entity, to model the symbol separation characteristics of the constraint, and a general switching function to tell how to satisfy the requirements of specific constraints with a given symbol separation characteristic.

Definitions 3.1 (Pseudo-dichotomy) Let \( S = \{x_0, x_1, \ldots, x_n\} \) be a set, the elements of which are called symbols, and \( B = \{0, 1\} \). A partition decomposition (PD) of \( S \) is a function \( \delta \) from \( S \times B \) to \( B \), such that \( \delta(x_0) = 0 \) and \( \delta(x_1) = 1 \). The binary relation \( \delta \) is called the satisfaction function of \( \delta \). The set \( \{0\} \) and \( \{1\} \) are called the 0-side and the 1-side of \( \delta \), respectively. A dichotomy of \( S \) is a PD \( \delta \) (\( x, y \) such that \( \delta(x) = \delta(y) = 0 \)).

A function \( \phi : \{0, 1\} \rightarrow \{0, 1\} \) is called a switching function of \( \delta \). A PD \( \delta \) is satisfied by \( x \) if \( \phi(\delta(x)) = 1 \).

A fixed pseudo-dichotomy (FPD) is a PD \( \delta \) (\( x, y \) such that \( \delta(x) = \delta(y) = 0 \)) where \( \delta \) is satisfied by \( x \).

A complete pseudo-dichotomy (CPD) is a PD \( \delta \) (\( x, y \) such that \( \delta(x) = \delta(y) = 0 \)).

A PD \( \delta \) is satisfied by \( x \) if \( \phi(\delta(x)) = 1 \).

We represent PDs using the value vector notation, which we employ to characterize binary relation graphs, instead of discrete functions on binary sets.

3.2 Constraint Classes
The encoding problems cited in this work can be expressed by a few constraint classes: local constraints, which can be input constraints, and distance-2 constraints, subdivided into output dominance, output disjunctive and compatibility constraints.

Local constraints express conditions that must be met in one or a subset of columns of the encoding. Input constraints were presented in Example 2.1. We note an encoding constraint by a pair \((x_1, x_2)\), where \((x_1)\) is the set of symbols whose codes must belong to the satisfying set of a cube that do not contain the codes of any symbol inside \((x_2)\). If \((x_1) \cup (x_2) = S\), the constraint is called full. If the cardinality of either \((x_1)\) or \((x_2)\) is 1 the constraint is called elementary. To satisfy one such constraint one encoding column suffices. Distance-2 constraints were proposed in [5] to guarantee fully stuck-at testable state assignment for FSMs. Such constraints are satisfied only if a Hamming distance of 2 is obtained between the codes of the symbols involved in it. This implies that at least two columns of the encoding have to be considered to achieve their satisfaction. We note such a constraint by \((x_1, x_2)\) for two symbols \(s_1\) and \(s_2\) that must be encoded with distance 2.

Global constraints must be verified by every column of the encoding. A global constraint \((x_1, x_2)\) is satisfied by \(x_2\) in every column where \(x_2\) is different from 0 it assumes the same value as \(x_1\). This is a dominator constraint between two symbols \(x_1\) and \(x_2\), and we note it by \((x_1, x_2)\). A disjunctive constraint, on the other hand, involves three symbols, \(x_1, x_2,\) and \(x_3\), where one of them, e.g., \(x_1\), is required to have a code that is the Boolean disjunction of the codes of the other two symbols, which is noted \((x_1, x_2, x_3)\).

Dominance and disjunctive constraints are found in the context of output encoding of combinational circuits, as well as in the state assignment of FSMs. A correspondence between \((x_1, x_2)\) and \((x_1, x_2)\) states that in no column of \(x_1\) of one set \(x_1\) and \(x_2\) can be 0 while the other is 1, noted \((x_1, x_2)\). This constraint has been identified in [2] and derives from the consideration of the state minimization problem during encoding.

3.3 The PD Unified Framework
We propose the organization of PDs into a framework capable of representing all conditions to be satisfied by the encoding.

Definition 3.2 (PD framework) Consider an algebraic structure \( F = (\{F_1, F_2\}, \), where \( F_1 \) is a set of pairs, where each pair has as first element a PD on a set \( S \) of symbols and as second element a positive integer, i.e., \( F_1 = \{([0, 1], 3)\}, F_2, F_3 \) is a set of PDs on \( S \). An encoding \( \xi \) of \( S \) satisfies \( F_1 \) if each \( \xi \) in \( F_1 \) is satisfied by at least as many columns of \( S \) as \( 0 \) and each \( \xi \) in \( F_2 \) is satisfied by every column of \( S \). If an encoding that satisfies \( F_1 \) exists, \( F_2 \) is called a PD framework of \( S \) and \( F_1 \) and \( F_2 \) are called the local part and the global part of \( F \), respectively.

From the definition of the PD framework we see that the local part expresses conditions that are independent of an encoding \( \xi \) of \( S \) while the global part collects conditions that need to be verified by every column of \( S \). The definition is not dependent upon the specific problem we are trying to solve, being applicable to a wide range of problems. Assuming that we do not consider PDs where the satisfaction function \( \xi(x) = 0 \) for every boolean vector \( x \), a special case of algebraic structure \( F = (F_1, F_2) \), where \( F_2 = \emptyset \) is always a PD framework, because an encoding can always be found that satisfies the local part alone. The reason for this is that PDs in the local part need to be satisfied always in one finite number of columns of the encoding. Thus, we can simply add columns to the encoding until all PDs are satisfied.

Once a PD framework is defined only if an encoding that satisfies it exists, establishing the framework is a task dependent on a constraint feasibility analysis, which in turn depends on the specific encoding problem at hand.

4 Mapping Constraints into PDs
Some works have suggested general formalisms for constrained encoding problems [9, 10]. Constrained encoding can benefit from the identification of compatibility classes inside the starting symbol set. Moreover, identifying these classes, we can apply encoding functions \( A \) that are injective and whose images are subsets of \( P(B^n) \) containing singletons only. Our approach does not modify the B&E problem in full extent, but, for instance, it is less restrictive than any other method found in the literature.

4.1 Representing Local Constraints
The input constraints generated by symbolic minimisation have the form of full input constraints [4]. For instance, let \( S = \)

704
(a, b, c, d, e, f, g, h, i) be the state set of some finite state machine, and let the pair \((a, b, c), (d, e, f, g, h, i)\) be one such full input constraint extracted from a symbolically mimimized cube table of some FSM. To model such constraints with PDSs, we simply choose \(\theta = \{a, b, c\}\) to represent \(P\), \(\theta = \{d, e, f, g, h, i\}\) to represent \(M\), and evaluate \(t\) to 1 only for columns \(O\) separating the codes of every two states in opposite sides of the PD. In this case, \(t\) is the disjunction of the two minterms \(a b c f g h i\) and \(a b c f g h i\) and is encoded as \(\text{SM} = 8 \text{SM} \neq \text{SM}\). Cubes having \(a_i = 0\) if \(p(c, f, g, h, i)\) are called non-conflict cubes, and \(t\) evaluating to 0 only for columns \(O\) separating the codes of every two states in opposite sides of the PD. In this case, \(t\) is the disjunction of the two minterms \(a b c f g h i\) and \(a b c f g h i\) appears in the encoding \(E\). It is clear that it takes one single column of \(S\) to satisfy this PD alone. This is sufficient, but not necessary to satisfy the full input constraint. To alleviate the restrictions imposed on \(S\), we may instead use the corresponding elementary input constraint in \(\theta\). In this case, the full constraint is produced by the SPDs \((a, b, c), \{f\}, \{g, h, i\}\) and \((a, b, c), \{f\}, \{g, h, i\}\) and \((a, b, c)\) and \((a, b, c)\). These SPDs may each be satisfied by a single full constraint, and the disjunction of these functions is defined in the same way it was defined for the full input constraint, and is the disjunction of a set of cubes which can be satisfied with more code possibilities than the original satisfaction function.

Distance-2 constraints are represented by PDSs in the same way as input constraints, with the possible exception that the cardinality function is different. Theorem 1 allows for this change in the cardinality function when no new full constraints are introduced.

### 4.2 Representing Global Constraints

Given two symbols \(a\) and \(b\), a dominance constraint \((a, b)\) can be translated into one single SPD \(\theta = \{a, b\}\) with \(p = \{a\}, \{b\}\), and the satisfaction function \(t = a \land b\).

Given three symbols \(a, b, c\), a disjunctive constraint \((a, b, c)\) can be expressed by three distinct SPDs \(\theta = \{a\}, \{b\}, \{c\}\), \(t = a \lor b \lor c\).

A compatibility constraint \((a, b, c)\) is modeled by one pseudo-dichotomy \(\theta = \{a\}, \{b\}, \{c\}\), \(t = a \lor b \lor c\).

### 5 A Case Study

Traditionally, state minimization (SM) and state assignment (SA) are separate procedures of sequential logic synthesis, but using such a strategy may prevent the obtainment of optimal state assignments [7]. To illustrate encodings where the symbol set may contain compatibility classes, we have chosen the problem of assigning codes to states of a FSM such that state minimization is taken in account during the encoding process, in what we call a simultaneous strategy. No theoretical findings on the relationship between the SM and SA problems has been provided in previous works [1, 8]. The method in [6] is feasible only for very small machines. The method proposed in [1] is reasonably efficient for machines with no less than 30 states, but the results are poorer than those obtained with a sequential strategy proposed in the same work.

### 5.1 Relationship between SM and SA

In [2], a formal relationship between the SM and SA problems was established, generating the results that the proofs can be found in [2] and are omitted here due to lack of space.

**Theorem 5.1 (Covered closed encoding)**

Let \(A = (I, S, O, \lambda, \mu)\) be an FSM and \(\mu\) be a closed coverage of constraints of \(A\). Build a functional injective encoding \(e: n \rightarrow \mathbb{N}^+, n \geq \log_2 |I| \times |S|\) and then use the closed encoding \(e(S) \rightarrow \mathbb{N}^+, \text{such that} \forall s, e(s) = e(k) \implies e(n, k) \in \mu\). Then, \(e\) is a valid state encoding of \(A\).

**Theorem 5.1** shows that it is possible to build valid encodings for the states of an FSM such that their length depends on the cardinality of the closed coverage of state compatibility classes, and not on the cardinality of the set of states. It also states that we may use non-injective, non-functional encodings to create the state compatibility class structure of the set \(S\), if any exists.

### 5.2 Incompatibility constraints non-violation

**Theorem 5.2**

**Theorem 5.2 (Incompatibility constraints non-violation)**

Given a finite state machine \(A = (I, S, O, \lambda, \mu)\), if two states \(s, t \in S\) are incompatibility constraints, then the set of full input constraint codes generated by symbolic minimization of a cube table of \(A\) assigns disjoint codes to \(s\) and \(t\).

**Theorem 5.3 (Closure constraints violation)**

**Theorem 5.3**

**Theorem 5.3**

The last two Theorems relate the SA input constraints with constraints arising from the SM problem. The first of them guarantees that all codes that have to be disjoint because of their incompatibility, are made disjoint by simply satisfying all input constraints obtained by symbolic minimization. The last Theorem, on the other hand, shows the undesirable effect arising from the satisfaction of all input constraints obtained by symbolic minimization, which is the assignment of disjoint codes to some otherwise compatible pairs of states. To avoid this effect during the encoding of states, we propose an original method of relaxation for the input constraints.

**Method 5.1 (Input constraints relaxation)**

Let \(A = (I, S, O, \lambda, \mu)\) be an FSM, with \(\mu\) being the set of compatible constraints of \(A\), and \(\theta\) a set of elementary input constraints of \(S\). Then,

1. For each pair \((s_i, s_j)\) do
2. If \(s_i = s_j = \emptyset\) or \((s_i, s_j) \in \theta\), such that \(s_i = (s_i)
3. Then, eliminate \((s_i, s_j)\) from \(\theta\).

The objective of the relaxation method for input constraints is to avoid encoding conditionally compatible states with disjoint codes. The correctness of the procedure is established by the following theorem.

**Theorem 5.4 (Input constraints relaxation)**

Given a finite state machine \(A = (I, S, O, \lambda, \mu)\), the set of all compatibility constraints \(\theta\) of \(A\), and a set of elementary input constraints \(\theta\) of \(S\), such that \(\emptyset\) is the result of the decomposition of all full face elementary constraints arising from symbolic minimization.

Theorem 5.4 states that the input constraints relaxation method to obtain a set of relaxed constraints \(\theta\) that respects all pairwise input constraints arising from symbolic minimization, while minimizing the number of compatibility constraints in \(\theta\), is a valid state assignment of \(A\).

**Corollary 5.1 (Bounds Preservation)**

The input constraints relaxation method does not increase the upper bound on the row cardinality of the encoded cube table predicted by symbolic minimization.

**Theorem 5.5**

Theorem 5.5 states that after applying the relaxation method it is still possible to find an encoding that preserves the input/output behavior of the original machine and is based on the relaxed constraints.

5.2 A PD Framework Encoding Method

Using the theoretical findings of the last Section, we propose a state encoding method considering state minimization. The method supports the use of all constraint classes mentioned in this work, although the implementation is limited today to considering input and compatibility constraints only. A PD framework is obtained as always. Starting with the input constraints generated by symbolic minimization, we decompose these into elementary input constraints with removal of duplicated constraints. The method proceeds by relaxing the input constraints using for this the constraints derived from an arbitrary state pair compatibility analysis. The relaxed constraint set, together with the compatibility constraints form a feasible set of constraints [2], which is mapped into a unified PD framework.

In [8], the PD framework is established below. From this, the PD framework is constructed.

The idea of the ASSTUCE method is to generate one column of the encoding at a time, so that the columns generated satisfies a maximum number of PDs in the local part of the unified framework, and do not violate any PD in the global part. After each column generation step, all satisfied PDs in the local part have the associated integer values incremented. For each value resulting 0 after this operation the associated PDs are accordingly eliminated, and column generation proceeds. There are two possible stop conditions, depending on what constrained encoding approach is chosen, complete or partial. If the state constraints encode columns 1, the method execution stops only when the local part is empty. Otherwise, execution stops if either the local part is empty, or if every incompatible pair of states is assigned disjoint codes.

The final encoding is the collection of generated bit columns. The complexity of the ASSTUCE method is bounded by \(O(n^2 + c)\), where \(n\) is the number of symbols (i.e., states in the FSM) and \(c\) is the number of non-don't care components in the initial PD matrix, which is bounded by the product of the number of symbols by the cardinality of the initial set of PDs, this last being proportional to the number of constraints.

### 5.3 Benchmark Results

The ASSTUCE method has been implemented as a computer program and compared against a strategic approach where state minimization is performed using the program STAMINA [6] and state assignment is done with the program NOVA [12]. The FSM test set used is part of the MCNC benchmarks. Our prototype implementation does not
consider output constraints yet. The program NOVA was accordingly parameterised to avoid their consideration (with the run-time option -lb), to a large extent. All the compared algorithms, in general, are solutions to the BCGE problem, but comparisons involving solutions to the CBC problem are also available in [2].

We divided the benchmarks into two groups, according to the presence or absence of non-trivial compatible state pairs in the original description. All comparison parameters are extracted from the minimal-combination part of the encoded FSM. The programs NOVA and ASTUCE rely on the ESPRESSO program to perform the combinational part minimisation after encoding. The same statement is true for the input constraints generation step. In this way, the comparisons reflect the differences arising from the encoding strategy alone. The data resulting from comparing ASTUCE and the partial encoding serial strategy based on the STAMINA and NOVA programs is presented in Table 1, for the benchmarks with at least one pair of compatible distinct states. Results for the other machines are discussed in [3].

ASTUCE and the partial encoding serial strategy based on NOVA are comparable for most parameters, with the serial strategy obtaining slightly better area results and ASTUCE obtaining slightly sparser machines but with reduced number of transistors in it, and less product terms. The consequences of these differences is that we judge the ASTUCE results more adapted to power dissipation issues in big PLAs, because of the combined effect of smaller areas corresponding to sparser PLAs. Besides, we know that sparser PLAs favor the use of topological optimisation tools during the low level synthesis of the FSM.

The advantages related to ASTUCE are a consequence of using non-linear, non-addictive encodings. In this way, cube merging is facilitated during the logic minimisation step, and even if the encoding length is increased, the final result may combine smaller areas with less dissipative power. However, the main issue here is to show that the formulation of the more general BCGE problem does not imply less efficient solutions for encoding problems, which validates the basic ideas of searching for more powerful encoding methods.

6 Conclusions and Future Work

The BCGE problem formulation was showed here to be a general approach to constrained encoding, which nonetheless does not imply less efficient implementations of encoding algorithms. The original formal statement for the pseudo-dichotomic concept allows that standard constraint satisfaction methods based on the discrete functions theory be applied to solve various VLSI design problems. The encoding problem case study showed the limitations of previous generic problem formulations to constrained encoding and stressed the potential benefits of using our proposal. We expect that this work will have a major impact on the way encoding problems are solved. Symbols generated by our approach are by construction sparser than those obtained with traditional encoding methods, as a result of employing non-linear, non-addictive encodings. The compactness of the results obtained so far with our prototype implementation indicates that we may achieve gains in power dissipation and communication complexity without compromising the area occupied by the circuit if more elaborate encoding schemes are developed.

We envisage the evolution of the present work in several directions. The first of these is to further validate the approach proposed here by obtaining more examples of encoding problems where the identification of equivalence and/or compatibility classes is fundamental to the search of optimal solutions. Second, we are presently doing research on the application of recently developed techniques for the manipulation of implicit representations of switching functions with the use of reduced ordered binary decision diagrams. We are also considering the application of these techniques to the representation and satisfaction of our PD framework. Third, we are interested in overcoming the limitation of using cube encodings. The eventual use of the Boolean relation concept may help in this task. Finally, we are considering the application of the formal paradigm developed here to sequential logic synthesis problems for FPGAs.

References


