Boolean Constrained Encoding: a new formulation and a case study

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

Abstract[1]

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 description abstraction levels. Our approach can capture equivalence and/or compatibility classes in the original symbol set to encode, by allowing symbols codes to be cubes of a Boolean space, instead of the usual minterms. Besides, we introduce a unified framework to represent encoding 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 consideration of state minimization. The practical 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 primarily 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 objective *optimization criteria*, 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 a set of 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 be useful for the state assignment problem alone [4], it represents a severe limitation if more powerful encoding strategies are required. 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 this characteristic, 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 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 formal statement which encompasses previously proposed formulations [9, 10], and which additionally allows that compatibility classes present in the symbol set be captured in the encoding process. Aiming 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, together with their use to solve 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/or 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 found less often in the scope of these problems are analysed 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 this 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 minimization. Section 5 also introduces a computer program implementation for solving the case study problem, and presents benchmark results. Finally, Section 6 lists a set of conclusions, and points directions for future work.

2 Basic Definitions and the BCE Problem

Definition 2.1 (Partition) *Given two sets S and T, a binary relation $\langle S,T,\pi\rangle$ is a* **partition** *of T iff it is onto and one-to-one. The image $\pi(s)$ of an element s of S is a* **block** *of partition π.*

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

Definition 2.2 (Lattice exponentiation function) *Let $\mathbf{X}$ denote the vector $(x_{n-1},\ldots,x_1,x_0)$ of variables taking values on S; given a variable $x_i \in \mathbf{X}$ and a subset $C_i \subset S_i$, we define the* **lattice exponentiation function** *as the discrete complete function $x_i^{(C_i)} : S_i \longrightarrow L$, where*

$$x_i^{(C_i)} = \begin{cases} r-1 & \textit{iff } x_i \in C_i \\ 0 & \textit{otherwise.} \end{cases}$$

Definitions 2.3 (Cube function) *A* **cube function** *or simply a* **cube** *is a discrete complete function $c : S \longrightarrow L$, where the values $c(\mathbf{X})$ are computed by the expression $c(\mathbf{X}) = l \wedge \bigwedge_{i=n-1}^{0} x_i^{(C_i)}$, with $l \in L$ and $C_i \in S_i$.*

The lattice element l is called the **weight** *of the cube. If $c(\mathbf{X})$ is such that for all C_i, $|C_i| = 1$, then $c(\mathbf{X})$ is a* **minterm**. *Given two cubes, $c(\mathbf{X}) = l \wedge \bigwedge_{i=n-1}^{0} x_i^{(C_i)}$, $l \in L, C_i \in S_i$, and $d(\mathbf{X}) = m \wedge \bigwedge_{i=n-1}^{0} x_i^{(D_i)}$, $m \in L, D_i \in S_i$, their* **supercube** *is a cube defined as $p(\mathbf{X}) = (l \wedge m) \wedge \bigwedge_{i=n-1}^{0} x_i^{(C_i \cup D_i)}$.*

The supercube definition is immediately extendible to sets of cubes with cardinality bigger than two. The cubes $c(\mathbf{X})$ and $d(\mathbf{X})$ are **disjoint** *if there is no $\mathbf{V} \in S$ for which $c(\mathbf{V}) = l$ and $d(\mathbf{V}) = m$, or if $l = 0$ or $m = 0$. The* **size** *of a cube $c(\mathbf{X})$ is $|\{\mathbf{X} \mid c(\mathbf{X}) = l\}|$, if $l \neq 0$, otherwise the size is 0. The* **satisfying set** *of c is the set of Boolean vectors $\{\mathbf{X} \mid c(\mathbf{X}) = l\}$, if $l \neq 0$, otherwise it is the empty set. Every element in this set is said to* **satisfy** *the cube function c. A* **switching cube function** *is a cube whose domain is $S = \mathcal{B}^n$ and whose codomain is $L = \mathcal{B}$, for some integer n.*

The usual definition of 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 $e : S \longrightarrow \mathcal{P}(\mathcal{B}^n)$, where $\mathcal{P}(\mathcal{B}^n)$ stands for the powerset (the set of all subsets) of $\mathcal{B}^n$. The integer n is called the* **length** *of the assignment. For every $s \in S$, the image $e(s)$ is called the* **code** *of s. Two codes $e(s)$, $e(t)$ are* **disjoint** *iff $e(s) \cap e(t) = \emptyset$, otherwise the codes are* **intersecting**. *An assignment that is an injection and where codes are pairwise disjoint is an* **injective encoding**, *in which case $\forall (s,t \in S)$, $s \neq t \Longrightarrow e(s) \cap e(t) = \emptyset$. Any assignment that is not injective is called* **non-injective**.

[1] Work started during doctoral program at the Université Catholique de Louvain, in Louvain-la-Neuve, Belgium. Financially supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brasil, under scholarship grant 205411/88-6.

A **cube assignment** *or* **cube encoding** *of S is an assignment e of S where every code $e(s)$ is the satisfying set of some cube, i.e. there is a switching cube $c(\mathbf{X})$ over $\mathcal{B}^n$ such that $c(\mathbf{X}) = 1 \Leftrightarrow \mathbf{X} \in e(s)$. Let the codes of a cube assignment e be represented as three-valued vectors $\mathbf{V} = v_{n-1} \ldots v_0$, and the cardinality of S be q. The three-valued column vectors obtained by selecting all elements v_i for some i, over all codes $e(s), s \in S$, have the form $(v_{i,0} \ldots v_{i,q-1})^T$, and are called* **columns** *of the encoding. The exponent T 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 $e : S \longrightarrow \mathcal{B}^n$, 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 cube 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 $\mathbf{x} = x_{n-1} \ldots x_1 x_0$ to represent a tuple of the set $\mathcal{B}^n$, and call it a **Boolean vector**.

Definition 2.5 (Discrete function satisfaction) *Let $f : S \longrightarrow L$ be a discrete function and $f_s : \mathcal{B}^p \longrightarrow \mathcal{B}^q$ be a general switching function. We say that f_s* **satisfies** *f under the two assignments $\varphi : S \longrightarrow \mathcal{B}^p$ and $\psi : L \longrightarrow \mathcal{B}^q$ if these assignments are such that $\forall (s \in S)\ f(s) = l \Longrightarrow f_s(\varphi(s)) = \psi(l)$.*

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 $C = \{f_0, \ldots, f_{m-1}\}$, where the elements $f_i \in C$ are switching functions $f_i : \mathcal{B}^n \longrightarrow \mathcal{B}$. Associate with each f_i a positive real number $c(f_i)$ and a discrete function g_i such that $g_i : (\mathcal{B}^k)^n \longrightarrow \mathcal{B}$. We call the elements of S* **symbols**, *and the elements of C* **encoding constraints** *on the symbols of S. The number $c(f_i)$ is the* **gain** *of f_i, while g_i is the* **encoding constraint satisfaction function** *of f_i. A constraint f_i is* **satisfied** *by a set $E \subseteq (\mathcal{B}^k)^n$ iff there is an element $e \in E$ such that $g_i(e) = 1$. The satisfaction of a constraint f_i by E is indicated here, with a little notational abuse by $g_i(E) = 1$. The Boolean constrained encoding problem consists in finding a function $h : S \longrightarrow \mathcal{P}(\mathcal{B}^k)$, where $\mathcal{P}(\mathcal{B}^k)$ is the powerset of $\mathcal{B}^k$, and such that k is minimized, and the gain $c(h)$ is maximized, where $c(h)$ is defined as*

$$c(h) = \sum_{i=0}^{m-1} c(f_i) \cdot g_i(\times_{j=0}^{n-1}(h(s_j)))$$

Function h is denominated 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 minimization only, and not considering output constraints nor any other constraints. Let us express this problem as an instance of the BCE problem.

The set of symbols S is the set of states of an FSM, to be encoded. The set of encoding constraints C on the other hand, must be computed from the face embedding constraints obtained by symbolic minimization. Consider, for instance, the FSM $\mathcal{A} = \langle I, Q, O, \delta, \lambda \rangle$ where $I = O = \mathcal{B}$, $T = \{a, b, c, d\}$, and where δ (the transition function) and λ (the output function) are given by the flow table in Figure 2.1, where also appears the grouping of entries obtained by symbolic minimization.

	0	1
a	d,1	c,0
b	d,-	a,0
c	d,0	a,0
d	c,1	b,1

Figure 1: Flow table and input constraints for FSM $\mathcal{A}$

To each grouping corresponds a face embedding constraint. These are associated to the present states of each entry group in the Figure [4]. We have thus six constraints, but only four distinct ones, which are $(\{a, b\}, \{c, d\}), (\{a\}, \{b, c, d\}), (\{b, c\}, \{a, d\}), (\{d\}, \{a, b, c\})$.

Remember the interpretation of these constraints [4]: two symbols in opposite sides of a constraint must have codes differing in at least one column of the final encoding. The encoding constraints in the form of switching functions are then: $f_0 = ab\bar{c}\bar{d} \vee \bar{a}\bar{b}cd$, $f_1 = a\bar{b}\bar{c}\bar{d} \vee \bar{a}bcd$, $f_2 = a\bar{b}\bar{c}d \vee \bar{a}bc\bar{d}$, and $f_3 = abc\bar{d} \vee \bar{a}\bar{b}\bar{c}d$. Indeed, suppose that states are encoded in the order $abcd$. To separate states c, d from states a, b (according to the first constraint), we need to have an encoding column that is either $(0011)^T$ or $(1100)^T$ in the solution. If a function f_i evaluate to 1, the constraint is respected. Then, the set C of the BCE problem is simply the set of all f_is.

Now, consider the modeling of the f_i gains $c(f_i)$ and the satisfaction functions g_i. To compute the gains, we observe that some constraints are repeated in the cube table obtained by symbolic minimization. We thus assign to each encoding constraint f_i a value $c(f_i)$ that is equal to the number of times the face embedding associated with f_i appears in the symbolically minimized cube table. This choice is justified by the fact that each entry set in the symbolically minimised cube table is associated with one row in the final minimized two-level implementation [12]. Then, satisfying a constraint guarantees that all groupings associated with this constraint can be performed in the final implementation. If not 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 minimization.

The constraint satisfaction function g_i, in this example, has an expression which is identical to the corresponding function f_i, 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 the corresponding function f_i is respected in *every* column of the constraint, like code compatibility constraints [2] or output constraints [3]. There are also cases where the f_i need to be satisfied in a specific number of encoding columns [5]. That is why the domain of the satisfaction functions g_i are all possible encodings of length k. Given this mapping, the state assignment problem can be reduced to the general BCE problem, and we need to look for an optimum encoding h of the state set Q, such that k is the minimum possible and $c(h)$ is maximum. ∎

Let us now give general interpretations for the elements in the BCE problem statement. S is a set of symbols to be encoded according to the constraints in C. The encoding constraints f_i, on the other hand, map a Boolean vector with the same cardinality as the set of symbols into 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_i the problem statement associates g_i, a function that characterizes the encoding constraint. It is through g_i that the behavior of the distinct encoding constraint classes can be accounted for. The encoding constraint satisfaction function g_i tells if the encoding constraint f_i 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 the major generalization 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 $g_i(\times_{j=0}^{n-1}(h(s_j)))$, evaluates to 1 iff the constraint f_i is satisfied by the final encoding. Otherwise, it evaluates to 0.

Finding function h is an NP-hard problem, since BCE is a generalisation of the state encoding problem [2]. The solution of the BCE problem in the general case is not unique. In most practical instances of the Boolean constrained encoding problem, the optimum solution is found only when considering a trade-off between the goals of minimising k and maximising the gain $c(h)$. Besides, for h to exist, the set of constraints C must be *feasible*. Constraint set feasibility is not treated here, and is also a very complex problem.

All proposals we could find in the available literature on encoding in VLSI design choose to solve restricted versions of the BCE problem. All such restricted versions can be put into two major classes [12].

Definitions 2.6 (Complete and partial constrained encoding) *Choose to satisfy all encoding constraints unconditionally, thus maximizing $c(h)$. At the same time, look for an encoding that minimizes k. This restricted version is called* **Complete (Boolean) Constrained Encoding** (CBCE). *Another restricted version of the Boolean constrained encoding problem is obtained as follows: establish a value for k (often the minimum possible), looking then for an encoding with length k that maximizes $c(h)$. 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 most kinds of constraints found in encoding problems at several levels of abstraction in VLSI descriptions. The widely accepted definition of (pseudo-)dichotomies is not as general as the definition of encoding constraints, and that is one of the reasons why we provide a more general definition of the former. On the other hand, the functions g_i account for the satisfaction of the constraints f_i across an arbitrary set of columns of the encoding. This justifies the proposal of a framework considering the effect of constraints that influence the composition of more than one column of the final encoding.

3.1 The Pseudo-Dichotomy Concept

Pseudo-dichotomies had originally little or no algebraic structure associated to it. No addition or product of dichotomies can be defined, although *ad hoc* operations for combining and splitting them, as well as concepts like compatibility and covering between dichotomies have found their way in previous publications [11, 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 useful to model single constraints in Boolean encoding problems. In general, these constraints consist on indications to make the codes of symbols disjoint or intersecting. Two-block partitions are adequate to model such a behavior. A two-block partition is interpreted as an indication to make bits on one block distinct from the bits in the other block, thus making the codes of symbols in distinct blocks disjoint. However, the partition definition implies that all elements of the symbol set be contained in some 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 characteristic 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 $S = \{s_0, \ldots, s_{n-1}\}$ be a set, the elements of which are called* **symbols**, *and $\mathcal{B} = \{0, 1\}$. A* **pseudo-dichotomy** (PD) *of S is an algebraic structure $\partial = \langle p, t\rangle$ where p is the graph of a binary relation $\langle \mathcal{B}, S, p\rangle$, with $p : \mathcal{B} \longrightarrow S$, such that $p(0) \cap p(1) = \emptyset$, and t is a switching function $t : \mathcal{B}^n \longrightarrow \mathcal{B}$. Function t is called the* **satisfaction function** *of ∂. The sets $p(0)$ and $p(1)$ are called the* **0–side** *and the* **1–side** *of ∂, respectively. A* **dichotomy** *of S is a PD $\partial = \langle p, t\rangle$ such that $p(0) \cup p(1) = S$. The binary relation with graph p is, in this case, a partition. A* **seed pseudo-dichotomy** (SPD) *is a PD $\partial = \langle p, t\rangle$ where either $\mid p(0) \mid = 1$ or $\mid p(1) \mid = 1$.*

Given a Boolean vector $\mathbf{X} = x_{n-1} \ldots x_0$, a PD ∂ is **satisfied** *by $\mathbf{X}$ iff $t(x_{n-1}, \ldots, x_0) = 1$. A* **flexible pseudo-dichotomy** *is a PD where $\forall x_{n-1} \ldots x_0 \in \mathcal{B}^n$, $t(x_{n-1}, \ldots, x_0) = t(\overline{x_{n-1}}, \ldots, \overline{x_0})$.*

A **fixed pseudo-dichotomy** (FPD) *is a PD $\partial = \langle p, t\rangle$ where the satisfaction function t is the cube function whose three-valued cube switching representation* [2] *for each position x_i is 0 if $s_i \in p(0)$, is 1 if $s_i \in p(1)$, and is - otherwise.*

Two PDs $\partial_1 = \langle p_1, t_1\rangle$, $\partial_2 = \langle p_2, t_2\rangle$ of S are **compatible** *iff there is at least one Boolean vector $\mathbf{X} = x_{n-1} \ldots x_0$ such that $t_1(\mathbf{X}) = t_2(\mathbf{X}) = 1$. A set of PDs is* **compatible** *if every two PDs in it are compatible. The PD ∂_1* **covers** *∂_2 iff $\forall \mathbf{X} \in \mathcal{B}^n$, $t_2(\mathbf{X}) = 1 \Rightarrow t_1(\mathbf{X}) = 1$. A set of PDs Δ* **covers** *a PD ∂ iff it contains a PD that covers ∂.*

We represent PDs using the *value vector* [2] notation, which we employ to characterize binary relation graphs, instead of discrete functions only. Given a PD $\partial = \langle p, t\rangle$ on the set of symbols S, ∂ may be described by the value vector $[p(0)\ p(1)]$, which contains the images of the elements 0 and 1 by the binary relation whose graph is p.

As an illustration, the face embedding constraints of Example 2.1 can be represented by the following set of PDs:

$$
\begin{aligned}
\partial_1 &= \langle p_1 = [\{c, d\}\{a, b\}], f_0 = ab\bar{c}\bar{d} \vee \bar{a}\bar{b}cd\rangle;\\
\partial_2 &= \langle p_2 = [\{b, c, d\}\{a\}], f_1 = a\bar{b}\bar{c}\bar{d} \vee \bar{a}bcd\rangle;\\
\partial_3 &= \langle p_3 = [\{a, d\}\{b, c\}], f_2 = a\bar{b}\bar{c}d \vee \bar{a}bc\bar{d}\rangle;\\
\partial_4 &= \langle p_4 = [\{a, b, c\}\{d\}], f_3 = abc\bar{d} \vee \bar{a}\bar{b}\bar{c}d\rangle.
\end{aligned}
$$

The structure of the satisfaction functions t_i is determined by the kind of constraint, face embedding or input constraints in this case. In a dichotomy $\partial = \langle p, t\rangle$, p is the graph of a partition of S. In a PD, p is the graph of a partition of a *subset* of S.

3.1.1 Generality of the PD Definition

The constraints a cube encoding of a set of symbols must satisfy are relationships among the symbols. They must be expressed as conditions to be respected among the symbols in some subset of columns of the encoding. PDs were defined to model each of these columns.

Given a generic PD $\partial = \langle p, t\rangle$ of a set S, the domain of a satisfaction function t is the set of all binary assignments of length 1 to symbols in S. Function t evaluates to 1 for every binary assignment that satisfies the PD, otherwise it evaluates to 0. In this way, PDs with a same binary relation p can be satisfied in different ways, if their satisfaction functions t are distinct. Thus, t allows that different kinds of constraints be accounted for.

The satisfaction function is not present in previous definitions of the PD concept. Actually, the first published applications have dealt with just one kind of constraint at a time [11]. As the need to manipulate other constraint kinds arose, the proposal of *ad hoc* frameworks took place to deal with the *anomalous* behavior of the new constraints [3]. Breaking the PD concept into an encoding part p and a satisfaction part t is a more general approach. Also, the consideration of new constraint kinds using our PD definition is straightforward. It suffices to define the conditions under which such a constraint is satisfied, generating a new type of function t.

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 or distance-2 constraints; and global 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 $(\{s_i\}, \{s_j\})$, where $\{s_i\}$ 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 $\{s_j\}$. If $\{s_i\} \cup \{s_j\} = S$, the constraint is called **full**. If the cardinality of either $\{s_i\}$ or $\{s_j\}$ 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. One such constraint is 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 a pair $\{[s_i], [s_j]\}$ for two symbols s_i and s_j that must be encoded with distance 2.

Global constraints must be verified by every column of the encoding. Given two symbols, s_1 and s_2, and an encoding e, s_1 dominates s_2 iff in every column where $e(s_2)$ is different from 0 it assumes the same value as $e(s_1)$. This is what a dominance constraint between two symbols states, and we note it by (s_1, s_2). A disjunctive constraint, on the other hand, involves three symbols, s_1, s_2 and s_3, where one of them, e.g. s_1 is required to have a code that is the Boolean disjunction of the codes of the other two symbols, which is noted $(s_1, s_2 s_3)$. 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 compatibility constraint between s_1 and s_2 states that in no column of e one of $e(s_1)$, $e(s_2)$ can be 0 while the other is 1, noted $\{s_1, s_2\}$. 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 attained by the encoding.

Definition 3.2 (PD framework) *Consider an algebraic structure $\mathcal{F} = \langle F_l, F_g\rangle$, where F_l 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. $F_l = \{(\partial_i, c_i)\}$. F_g is a set of PDs on S. An encoding Ξ of S satisfies $\mathcal{F}$ iff each ∂_i in F_l is satisfied by at least as many columns of Ξ as c_i and each element in F_g is satisfied by every column of Ξ. If an encoding that satisfies $\mathcal{F}$ exists, $\mathcal{F}$ is called a* **PD framework** *of S, and F_l and F_g are called the* **local part** *and the* **global part** *of $\mathcal{F}$, respectively.*

From the definition of PD framework we see that the local part expresses conditions that need to occur in some subset of columns of an encoding Ξ of S, while the global part collects conditions that need to be verified by every column of Ξ. 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 $t(\mathbf{X}) = 0$ for every Boolean vector $\mathbf{X}$, a special case of algebraic structure $\mathcal{F} = \langle F_l, F_g\rangle$, where $F_g = \emptyset$, 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.

Since 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 formulations for constrained encoding problems [9, 10]. Constrained encoding can benefit from the identification of compatibility classes inside the starting symbol set. Most of the works do not allow identifying these classes, since they rely upon encoding functions h that are injective and whose images are subsets of $\mathcal{P}(\mathcal{B}^k)$ containing singletons only. Our approach does not model the BCE problem in its full extent either. However, it is less restrictive than any other method found in the literature.

4.1 Representing Local Constraints

The input constraints generated by symbolic minimization have the form of full input constraints [4]. For instance, let $S =$

$\{a, b, c, d, e, f, g, h, i\}$ 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 minimised cube table of some FSM. To model this constraint with PDs, we may choose $\theta = \langle p, t\rangle$ such that $p = [\{d, e, f, g, h, i\}, \{a, b, c\}]$ and t evaluating to 1 only for columns separating the codes of every two states in opposite sides of the PD. In this case, t is the disjunction of the two minterms $abc\overline{d}\overline{e}\overline{f}\overline{g}\overline{h}\overline{i}$ and $\overline{a}\overline{b}\overline{c}defghi$. Cubes having x_i if $s_i \in p(1)$ are called **direct cubes** of t, while the ones having $\overline{x_i}$ are the **reverse cubes** of t. This PD is satisfied iff one of the two column encodings 111000000^T or 000111111^T appears in an encoding Ξ. It is clear that it takes one single column of Ξ to satisfy this PD alone. This is sufficient, but not necessary to satisfy the full input constraint. To alleviate the restrictions imposed on Ξ, we may instead use the corresponding elementary input constraints in ϕ. In the example, the full input constraint would produce the SPDs $[\{d\}, \{a, b, c\}]$, $[\{e\}, \{a, b, c\}]$, $[\{f\}, \{a, b, c\}]$, $[\{g\}, \{a, b, c\}]$, $[\{h\}, \{a, b, c\}]$ and $[\{i\}, \{a, b, c\}]$. These SPDs may each be satisfied separately, along several columns of the encoding. The satisfaction function t 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 PDs in the same way as input constraints. The fact that they require satisfaction in exactly two columns of the encoding is accounted for through the local part of the framework, where the integer associated to input constraints is 1, while for distance-2 constraints it is 2.

4.2 Representing Global Constraints

Given two symbols a and b, a dominance constraint (a, b) can be translated into one single SPD $\theta = \langle p, t\rangle$, where $p = [\{a\}, \{b\}]$, with the satisfaction function $t = a \vee \overline{b}$.

Given three symbols a, b and c, a disjunctive constraint (a, bc) can be expressed by three distinct SPDs $\theta_1 = \langle p_1 = [\{a\}, \{b\}], t_1 = a \vee \overline{b}\rangle$, $\theta_2 = \langle p_2 = [\{a\}, \{c\}], t_2 = a \vee \overline{c}\rangle$, $\theta_3 = \langle p_3 = [\{b, c\}, \{a\}], t_3 = bc \vee \overline{a}\rangle$.

A compatibility constraint $\{a, b\}$ is modeled by one pseudo-dichotomy $\theta = \langle p = [\{a\}, \{b\}], t = ab \vee \overline{a}\overline{b}\rangle$.

5 A Case Study

Traditionally, state minimization (SM) and state assignment (SA) are separate procedures of sequential logic synthesis, but using such a *serial 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 an FSM such that state minimization is taken into 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 [8] is feasible only for very small machines. The method proposed in [1] is reasonably efficient for machines with no less than 30 states, but its results are poorer than those obtained with a serial 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 summarized below. Formal proofs can be found in [2] and are omitted here due to lack of space.

Theorem 5.1 (Closed cover encoding) *Let $\mathcal{A} = \langle I, S, O, \delta, \lambda\rangle$ be an FSM and κ be a closed cover of compatibles [6] of $\mathcal{A}$. Build a functional injective encoding $\epsilon : \kappa \longrightarrow \mathcal{B}^n$, with $n \geq \lceil \log_2 |\kappa| \rceil$ and then build the encoding $e : S \longrightarrow \mathcal{P}(\mathcal{B}^n)$, such that $\forall s \in S$, $e(s) = \{\epsilon(k) \,|\, k \in \kappa, s \in k\}$. Then, e is a valid state encoding of $\mathcal{A}$.*

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 cover 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 capture the state compatibility class structure of the set S, if any exists.

Theorem 5.2 (Incompatibility constraints non-violation) *Given a finite state machine $\mathcal{A} = \langle I, S, O, \delta, \lambda\rangle$, if two states $s, t \in S$ are incompatible, any valid state encoding that respects the whole set of full input constraints generated by symbolic minimization of a cube table describing $\mathcal{A}$ assigns disjoint codes to s and t.*

Theorem 5.3 (Closure constraints violation) *Given an FSM $\mathcal{A} = \langle I, S, O, \delta, \lambda\rangle$, if two states $s, t \in S$ are conditionally compatible, any encoding that respects the whole set of full input constraints generated by symbolic minimization of a cube table of $\mathcal{A}$, assigns disjoint codes to s and t.*

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 $\mathcal{A} = \langle I, S, O, \delta, \lambda\rangle$ be an FSM, with θ being the a set of compatibility constraints of $\mathcal{A}$, and ϕ a set of elementary input constraints of S. Then,

for each pair $(\{s_i\}, s_k) \in \phi$ **do**
  if $\exists(s_l, s_k) \in \theta$ **or** $(s_k, s_l) \in \theta$, such that $s_l \in \{s_i\}$
  then, eliminate s_l from $\{s_i\}$ in $(\{s_i\}, s_k) \in \phi$;
    if the resulting set $\{s_i\} = \emptyset$
    then, eliminate $(\{s_i\}, s_k)$ from ϕ;

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 and Corollary.

Theorem 5.4 (Input constraints relaxation) *Given a finite state machine $\mathcal{A} = \langle I, S, O, \delta, \lambda\rangle$, the set of all compatibility constraints θ of $\mathcal{A}$, and a set of elementary input constraints ϕ of S, suppose that ϕ is the result of the decomposition of all full face embedding constraints arising from symbolic minimization. Apply the input constraints relaxation method to ϕ, obtaining a set of relaxed input constraints ϕ'. Then, any state encoding Ξ' of $\mathcal{A}$ that respects all relaxed input constraints in ϕ' and all compatibility constraints in θ, is a valid state assignment of $\mathcal{A}$.*

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 5.4 ensures that after applying the relaxation method it is still possible to find an encoding that preserves the input/output behavior of the original machine based on the relaxed constraints. An interesting result arising from symbolic minimization is that it generates an upper bound for the row cardinality of a two-level implementation of the combinational part of the initial FSM [4]. One may question if this bound is still valid after applying relaxation to these constraints. Corollary 5.1 ensures that this is indeed the case.

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 follows. 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 ordinary 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.

Once the PD framework is established, the solution of our encoding problem can be found by any constraint satisfaction method applicable to the framework. We have developed the **ASSTUCE method**, which is in fact a generalization of an existing greedy heuristic technique to generate one encoding column at a time, proposed in [10]. The original method could not be used, since it is limited to functional encodings, and we needed to generate cube encodings.

The idea of the ASSTUCE method is to generate one column of the encoding at a time, so that the column 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 value decremented. 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 complete constrained encoding is chosen, 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 $\mathcal{O}(n^2 + c)$, where n is the number of symbols (i.e. states of 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 serial strategy 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 do not

Table 1: ASSTUCE versus partial encoding serial strategy for FSMs with non-trivial compatible pairs

FSM	a_area	n_area	sn_area	a_time	n_time	sn_time	a_pt	n_pt	sn_pt	a_cl_f	n_cl_f	sn_cl_f	a_tr	n_tr	sn_tr	a_spty	n_spty	sn_spty
s27	198	234	216	0.28	0.10	0.10	11	13	12	3	3	3	43	62	51	78.28	73.5	76.39
beecount	144	247	160	0.23	0.10	0.10	9	13	10	2	3	2	50	68	54	65.28	72.47	66.25
lion9	102	136	77	0.43	0.30	0.10	6	8	7	4	4	2	24	35	24	76.47	74.26	68.83
ex5	120	252	96	0.25	0.50	0.00	8	14	8	3	4	2	34	88	34	71.67	65.08	64.58
ex7	120	306	96	0.27	0.30	0.10	8	17	8	3	4	2	38	107	36	68.33	65.03	62.5
ex3	144	324	96	0.18	0.20	0.10	8	18	8	4	4	2	33	103	35	77.08	68.21	63.54
bbara	380	550	380	0.78	0.20	0.20	20	25	20	3	4	3	94	129	94	75.26	76.55	75.26
opus	304	448	448	1.05	0.20	0.10	18	16	16	4	4	4	139	128	123	72.42	71.43	72.54
train11	85	153	66	0.37	0.60	0.10	5	9	6	4	4	2	26	47	24	69.41	69.28	63.64
mark1	697	684	646	0.93	5.10	4.90	17	18	17	5	4	4	145	115	117	79.20	83.19	81.73
sse	972	990	1023	1.67	0.50	1.10	27	30	31	5	4	4	196	191	206	79.84	80.71	79.86
bbsse	972	990	1023	1.62	0.40	1.20	27	30	31	5	4	4	196	191	206	79.84	80.71	79.86
ex2	394	609	195	1.28	0.30	1.00	18	29	13	9	5	3	91	171	66	84.68	71.92	66.15
tma	1230	1155	1295	5.97	6.60	13.30	30	33	37	7	5	5	241	230	257	80.41	80.09	80.15
ex1	2288	2496	2132	5.37	6.30	5.00	44	48	41	5	5	5	399	422	330	82.56	83.09	84.52
tbk	1680	4620	1431	103.22	140.7	24.5	56	154	53	5	5	4	614	1423	553	63.45	69.2	61.36
scf	17420	18471	16244	603.68	105.8	59.8	130	141	124	8	7	7	1541	1469	1383	91.15	92.04	91.49
s298	16632	22464	10332	10637.90	828.8	266.6	308	624	287	14	8	8	3824	6783	2586	77.01	69.81	74.96

Prefixes:
a_ : results obtained by running ASSTUCE
n_ : results obtained by running NOVA alone
sn_ : results obtained by running STAMINA + NOVA

Suffixes:
area : area estimate of the minimized combinational part for the encoded FSM
pt : number of product terms in the minimized combinational part of the encoded FSM
cl_f : encoding length time : total execution time
tr : number of transistors in the minimized combinational part of the encoded FSM
spty : percentual sparsity of the minimized combinational part for the encoded FSM

consider output constraints yet. The program NOVA was accordingly parameterised to avoid their consideration (with the run-time option -e ih), to allow a fair comparison. Also, the compared strategies are solutions to the PBCE problem, but comparisons involving solutions to the CBCE 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 minimised two-level combinational part of the encoded FSM. The programs NOVA, and ASSTUCE 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 ASSTUCE and the partial encoding serial strategy based on the STAMINA and NOVA programs is depicted in Table 1, for the benchmarks with at least one pair of compatible distinct states. Results for the other machines are discussed in [2].

ASSTUCE and the partial encoding serial strategy based on NOVA are comparable for most parameters, with the serial strategy obtaining slightly better area results and ASSTUCE 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 ASSTUCE results more adapted to consider 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 optimization tools during the low level synthesis of the FSM.

The advantages related to ASSTUCE are a consequence of using non-functional, non-injective 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 dissipated power. However, the main issue here is to show that the formulation of the more general BCE problem does not imply less efficient solutions for encoding problems, which validates the basic idea of searching for more powerful encoding methods.

6 Conclusions and Future Work

The BCE 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-dichotomy 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 codes generated by our approach are by construction sparser than those obtained with traditional encoding methods, as a result of employing non-injective, non-functional encodings. The competitiveness 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

[1] M. Avedillo et al. "SMAS: a program for concurrent state reduction and state assignment of finite state machines". *ISCAS'91*, pp. 1781-1784.

[2] N. Calazans. *State minimization and state assignment of finite state machines: their relationship and their impact on the implementation*. PhD thesis, Université Catholique de Louvain, Belgium, 1993.

[3] M. Ciesielski et al. "A unified approach to input-output encoding for FSM state assignment". *DAC'91*, pp. 176-181.

[4] G. de Micheli et al. "Optimal state assignment for finite state machines". *ITCAD*, pp. 269-284, July 1985.

[5] S. Devadas et al. "A synthesis and optimization procedure for fully and easily testable sequential machines". *ITCAD*, pp. 1100-1107, October 1989.

[6] G. Hachtel et al. "Exact and heuristic algorithms for the minimization of incompletely specified state machines". *EDAC'91*, pp. 184-191.

[7] J. Hartmanis et al. "Some dangers in state reduction of sequential machines". *Information and Control*, pp. 252-260, September 1962.

[8] E. Lee et al. "Concurrent minimization and state assignment of finite state machines". *International Conference on Systems, Man and Cybernetics*, pp. 248-260, 1984.

[9] B. Lin et al. "A generalized approach to the constrained cubical embedding problem". *ICCD'89*, pp. 400-403.

[10] C. Shi et al. "Efficient constrained encoding for VLSI sequential logic synthesis". *EURO-DAC'92*, pp. 266-271.

[11] J. Tracey. "Internal state assignment for asynchronous sequential machines". *ITEC*, pp. 551-560, August 1966.

[12] T. Villa et al. "NOVA: state assignment of finite state machines for optimal two-level implementations". *ITCAD*, pp. 905-924, September 1990.