Encoding Multi-Valued Functions for Symmetry[*]

Ko-Lung Yuan[1], Chien-Yen Kuo[1], Jie-Hong R. Jiang[1,2], and Meng-Yen Li[3]

[1]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 10617, Taiwan
[2]Department of Electrical Engineering, National Taiwan University, Taipei 10617, Taiwan
[3]Information and Communications Research Lab., ITRI, Hsunchu 31040, Taiwan

ABSTRACT

In high-level designs, variables are often naturally represented in a symbolic multi-valued form. Binary encoding is an essential step in realizing these designs in Boolean circuits. This paper poses the encoding problem with the objective of maximizing the degree of symmetry, which has many useful applications in logic optimization, circuit rewiring, functional decomposition, etc. In fact, it is guaranteed that there exists a full symmetry encoding with respect to every input multi-valued variable for all multi-valued functions. We propose effective computation for finding such encoding by solving a system of subset-sum constraints. Experiments show unique benefits of symmetry encoding.

1. INTRODUCTION

Multi-valued (MV) logic is an extension of propositional logic with its domain containing more than two values. In high-level system design, often constants, variables, and functions are specified at the word level, and their domain values are naturally represented by multi-valued symbols. Nevertheless these symbols have to be binary encoded for final logic circuit realization. For example, the states of a finite state machine are pure symbols and can be encoded in binary codes for circuit implementation.

Optimization of a high-level multi-valued design can be performed in the following steps [3]. First, the design is optimized directly in the multi-valued logic domain [11, 15]. Second, an optimized multi-valued logic expression is then encoded into binary. Finally, conventional binary logic synthesis can then be performed on the Boolean netlist. For instance, the multi-valued logic synthesis tool MVSIS [10] has been developed to support such multi-valued design flow. Optimization at the multi-valued domain is advantageous due to the fact that some structural information is available only at multi-valued domain, and gets lost when translated into binary [13]. On the other hand, encoding multi-valued logic into binary may not be unique. It is well known that binary encoding may have drastic effects on final circuit complexity. For example, there has been extensive work on input, output, and state encodings for optimal two-level logic implementation [1, 8, 9, 19]. Encoding for multi-valued multi-level network was studied in [12].

Most prior work on multi-valued logic encoding was concerned about minimizing area in terms of the number of cubes and literals. In contrast, this paper formulates a new symmetry encoding problem. Given a multi-valued function, we intend to encode it into binary functions that exhibit a maximal degree of symmetry [17] on their input variables. In essence, a Boolean function is symmetric on two input variables if their swap does not change the functionality. Since symmetry is a functional property, it is an invariant and remains intact under logic transformation. Therefore, encoding using symmetry as an optimization metric can be more meaningful than using area, which is, after all, hard to predict under logic transformations.

Symmetric functions have several desirable properties. For example, symmetry may allow logic rewiring to reduce routing congestion [7, 5], may ease engineering change order (ECO) for late design changes, may simplify BDD representation [18], may reduce communication complexity in functional decomposition [16], may accelerate Boolean matching in technology mapping [20], and other benefits. Therefore encoding for symmetry is potentially good for many applications. Nevertheless, to the best of our knowledge, it has not been studied in the literature (except our recent work [14]).

As a motivating example, consider the modular arithmetic function $F : \mathbb{Z}_3 \times \mathbb{Z}_3 \to \mathbb{Z}_3 \times \mathbb{Z}_3$ with $F(X_1, X_2) = (X_1 + X_2, X_1 \times X_2)$ for the summation and multiplication operators being over integers modulo 3. By a naive encoding $0 \mapsto 00$, $1 \mapsto 01$, and $2 \mapsto 10$, the resultant four output Boolean functions (two for summation and two for multiplication) exhibit no symmetry. However, let $0 \mapsto 00$, $1 \mapsto 01$, and $2 \mapsto 11$, in the arithmetic system along with the following proper function value assignments

X_1 \ X_2	0: 00	1: 01	2: 11	3: 10
0: 00	(0,0)	(1,0)	(2,0)	(1,0)
1: 01	(1,0)	(2,1)	(0,2)	(2,1)
2: 11	(2,0)	(0,2)	(1,1)	(0,2)
3: 10	(1,0)	(2,1)	(0,2)	(-,-)

where symbol "3" denotes the unused code 10 and "-" denotes a don't care value. Any of the encoded four output Boolean functions is symmetric on the two Boolean variables of X_1 and on the two Boolean variables of X_2. A natural question is how to systematically find such codes and function value assignments.

This paper formulates the Symmetry Encoding Problem (SEP) in terms of solving a system of general subset-sum constraints. Theoretically, we show that 1) output encoding is irrelevant to symmetry and only input encoding matters, and 2) when input MV variables are treated independently, full symmetry can always be achieve within the Boolean variables associated to an MV variable by code length expansion. Practically, we propose an effective decision procedure for solving a system of subset-sum constraints, whose solution corresponding to symmetry encoding. Moreover, we propose an encoding method that provides a tradeoff between the degree of symmetry and the length of code for practical realization of MV functions. Experimental results show scalable

[*]This work was supported in part by National Science Council under grants NSC 99-2923-E-002-005-MY3, 100-2923-E-002-008, and 101-2923-E-002-015-MY2.

encoding of functions with tens of thousands of input values. The encoded Boolean functions exhibit a unique high degree of symmetry, whereas circuit area and logic depth can be maintained comparable to other optimal encoding methods.

The rest of this paper is organized as follows. Section 2 provides the preliminaries. Section 3 formulates the symmetry encoding problem as solving a system of subset-sum constraints. Section 4 proposes a decision procedure for solving subset-sum constraint. Experimental evaluation is presented in Section 5, and conclusions are drawn in Section 6.

2. PRELIMINARIES

In the sequel, $\mathbb{B}$, $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Z}_n$ represent the sets of truth values $\{0,1\}$, natural numbers, integers, and integers modulo n, respectively. Also, we shall use upper-case letters to denote MV functions, e.g., F, and MV variables, e.g., X; lower-case letters to denote binary functions, e.g., f_i, and binary variables, e.g., x_i.

2.1 Symmetry and Functions

The notion of symmetry for a Boolean functions and a Boolean function vector is defined as follows.

Definition 1. *A Boolean function $f : \mathbb{B}^m \to \mathbb{B}$ is* symmetric *on input variables x_i and x_j for $i \neq j$ if $f(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_m) = f(x_1, \ldots, x_j, \ldots, x_i, \ldots, x_m)$. In this case, x_i and x_j are said to form a* symmetric pair *in f. A Boolean function f is* fully symmetric *if every pair of input variables forms a symmetric pair.*

Definition 2. *A Boolean function vector $\vec{f} : \mathbb{B}^m \to \mathbb{B}^n$ is* symmetric *on input variables x_i and x_j for $i \neq j$ if, for* every *Boolean function of $f_1, \ldots, f_n$ in the vector, the two variables form a symmetric pair. Also, the vector is* fully symmetric *if all Boolean functions $f_1, \ldots, f_n$ are fully symmetric.*

The collection of all symmetric pairs of $\vec{f}$ forms a *symmetry relation*, which is an equivalence relation satisfying reflexivity (by viewing x_i and itself as a symmetric pair), symmetry, and transitivity.

Given an MV function $F : D_1 \times \ldots \times D_p \to C_1 \times \ldots \times C_q$, a *(binary) encoding* $\mathcal{E}$ maps every input value $d \in D_i$ for $i = 1, \ldots, p$ and output value $c \in C_j$ for $j = 1, \ldots, q$ into Boolean vector $\mathcal{E}(d) \in \mathbb{B}^{m_i}$ and vector $\mathcal{E}(c) \in \mathbb{B}^{n_j}$, respectively, such that distinct MV symbols remain distinguishable after encoding. Note that this paper assumes the encoding $\mathcal{E}$ is *injective*, where an MV symbol is mapped to only one binary code. Necessarily $m_i \geq \lceil \log_2 |D_i| \rceil$ and $n_j \geq \lceil \log_2 |C_j| \rceil$. Effectively $\mathcal{E}$ transforms F into a Boolean function vector $\vec{f} : \mathbb{B}^{m_1} \times \cdots \times \mathbb{B}^{m_p} \to \mathbb{B}^{n_1} \times \cdots \times \mathbb{B}^{n_q}$.

In this work, we are concerned with the symmetric pairs formed by the input variables of $\vec{f}$ that encode the same MV input variable of F. Therefore, an equivalence class of symmetric input variables, called a *symmetric class*, is local to their original MV input variable. To assess the degree of symmetry for a function vector $\vec{f}$ encoded from F, we use the following metric.

Definition 3. *The* symmetry degree *of $\vec{f}$ with respect to an MV input variable X_i of F is the cardinality of the largest symmetric class among the variables that encode X_i.*

Definition 4. *A* cofactor *of a Boolean function vector $\vec{f}(\vec{x})$ with respect to some of its inputs $\vec{x}' = b$ for $\vec{x}' \subseteq \vec{x}$ and $b \in \mathbb{B}^{|\vec{x}'|}$, denoted $\vec{f}|_{\vec{x}'=b}$, is a partial valuation of $\vec{f}$ with $\vec{x}'$ substituted with b. Similarly, a* cofactor *of an MV function $F(X_1, \ldots, X_p)$ with respect to its input $X_i = d$ for d a domain value of X_i, denoted $F|_{X_i=d}$, is a partial valuation of F with its input X_i substituted with d.*

Example 1. *Consider the modular arithmetic function $F : \mathbb{Z}_4 \times \mathbb{Z}_4 \to \mathbb{Z}_4 \times \mathbb{Z}_4$ with $F(X_1, X_2) = (X_1 + X_2, X_1 \times X_2)$ for X_1 and X_2 being integers modulo 4. Its function table is shown below.*

X_1 \ X_2	0	1	2	3
0	(0,0)	(1,0)	(2,0)	(3,0)
1	(1,0)	(2,1)	(3,2)	(0,3)
2	(2,0)	(3,2)	(0,0)	(1,2)
3	(3,0)	(0,3)	(1,2)	(2,1)

The cofactor of F with respect to $X_1 = 0$ is a new function with function values $(0,0)$, $(1,0)$, $(2,0)$, and $(3,0)$ for $X_2 = 0, 1, 2,$ and 3, respectively.

2.2 Multisets and Subset-sum Constraints

A (finite) *multiset* A of integers is a two-tuple (S, μ) consisting of a (finite) set $S \subseteq \mathbb{Z}$ and a multiplicity function $\mu : \mathbb{Z} \to \mathbb{N}$ with $\mu(e)$ denoting how many times an element $e \in \mathbb{Z}$ is present in A. (For $e \notin S$, we assume $\mu(e) = 0$.) That is, the notion of a multiset generalizes that of a set in its allowance of multiple occurrences of an element. For example, $A = (\{-2, 1, 2, 5\}, \{\mu(-2) = 1, \mu(1) = 2, \mu(2) = 1, \mu(5) = 1\})$ is a multiset, also represented as $\{-2, 1, 1, 2, 5\}$.

Given a multiset A of integers, the summation of all its member integers (counting the multiplicities) is denoted as $\sum(A)$ and the size of A, denoted $|A|$, can be computed by $\sum(\mu(a))$ where $a \in A$. For example, $\sum(A) = 7$ and $|A| = 5$ for $A = \{-2, 1, 1, 2, 5\}$. The *union of two multisets* $A = (S_A, \mu_A)$ and $B = (S_B, \mu_B)$, denoted $A \sqcup B$, is a multiset $(S_A \cup S_B, \mu_A + \mu_B)$ with $(\mu_A + \mu_B)(e) = \mu_A(e) + \mu_B(e)$. We say that a multiset $A = (S_A, \mu_A)$ is a *sub-multiset* of $B = (S_B, \mu_B)$, denoted $A \sqsubseteq B$, if $S_A \subseteq S_B$ and $\mu_A(e) \leq \mu_B(e)$ for every $e \in \mathbb{Z}$.

Given a multiset A of integers and an integer t, the *subset-sum problem* asks whether there is a multiset $A' \sqsubseteq A$ with $\sum(A') = t$. For example, $A = \{-2, 1, 1, 2, 5\}$ has a sub-multiset $\{-2, 1, 1\}$ that sums up to $t = 0$. As a matter of fact, the subset-sum problem is NP-complete [6]. In this paper, we generalize the subset-sum problem to a system of subset-sum constraints as follows.

Definition 5 (subset-sum constraint). *Given two multisets E and $T = \{t_1, \ldots, t_k\}$ of integers, the* subset-sum constraint problem of E summing to T *asks whether there exist $E^i \sqsubseteq E$ for $i = 1, \ldots, k$ such that $\sum(E^i) \bowtie t_i$ and $\bigsqcup E^i \sqsubseteq E$, where $\bowtie \in \{<, \leq, >, \geq, =\}$.*

Note that the standard subset-sum problem is the special case with $\bowtie$ set to $=$ and $|T| = 1$.

3. ENCODING FOR SYMMETRY

3.1 Problem Statement

Problem 1 (encoding for full symmetry). *Given an MV function $F(X_1, \ldots, X_p)$, we aim to find an encoding $\mathcal{E}$ such that the encoded Boolean function vector $\vec{f}$ is full symmetric within every set of Boolean variables that encode MV input X_i of F, for $i = 1, \ldots, p$.*

Problem 2 (encoding for partial symmetry). *Given an MV function $F(X_1, \ldots, X_p)$ and the coding lengths for $X_1, \ldots, X_p$, we aim to find an encoding $\mathcal{E}$ such that the encoded Boolean function vector $\vec{f}$ is maximally symmetric within every set of Boolean variables that encode MV input X_i of F, for $i = 1, \ldots, p$.*

By the following theorem, we know that the encoding for MV output values (referred to as output encoding) are irrelevant to the symmetry relation of $\vec{f}$ and only the encoding for MV input values (referred to as input encoding) matters.

THEOREM 1. *Given an MV function $F : D_1 \times \cdots \times D_p \rightarrow C_1 \times \cdots \times C_q$, let $\mathcal{E}_1$ and $\mathcal{E}_2$ be two (injective) encodings that differ only in output encoding. Then their respective encoded Boolean function vectors $\vec{f}(\vec{x})$ and $\vec{g}(\vec{x})$ exhibit the same symmetry relation.*

PROOF. Since the encodings $\mathcal{E}_1$ and $\mathcal{E}_2$ are injective, their left inverses exist. Let $\mathcal{E}_1^{-1}$ be the left inverse of $\mathcal{E}_1$. The composite function $\mathcal{E}_2 \circ \mathcal{E}_1^{-1}$ maps, for every output value $c \in C_i$ with $i = 1, \ldots, q$, the output code of $\vec{f}$ encoding c to the output code of $\vec{g}$ encoding the same c. Let u be a truth assignment to the variable vector $\vec{x}$ and u' be a permuted version of u with a swap of a pair of bits. Then $\vec{f}(u) = \vec{f}(u')$ if and only if $\vec{g}(u) = \vec{g}(u')$. Hence the symmetry relations of $\mathcal{E}_1$ and $\mathcal{E}_2$ are the same. ∎

Accordingly we may focus on input encoding, and even determine output encoding afterwards as it does not affect symmetry relation. This property is crucial for finite state machine state encoding, which involves both input and output encodings (essentially current states and next states should be consistently encoded). Therefore in what follows we focus on the input encoding.

3.2 Encoding for Full Symmetry

We present some notation before introducing the theory.

DEFINITION 6 (PATTERN AND PATTERN SET). *We call a Boolean vector $\vec{p} \in \mathbb{B}^m$ a* pattern *of length m. Let P^m denote the set of patterns of length m, and $P^m_h \subset P^m$ be the set of patterns with hamming weight h (the number of 1's in a binary code).*

So $|P^m| = 2^m$ and $|P^m_h| = \binom{m}{h}$. For example, for $P^3_1 = \{001, 010, 100\}$ and $P^2 = \{00, 01, 10, 11\}$, $|P^3_1| = \binom{3}{1} = 3$ and $|P^2| = 2^2 = 4$.

For (injective) symmetry encoding, we associate a unique pattern to each (MV) input symbol (recall the focus of input encoding). The problem of associating patterns to appropriate input symbols can be considered as solving a system of subset-sum constraints with $\bowtie$ fixed to $\geq$ as follows.

THEOREM 2. *Given an MV function $F(X_1, \ldots, X_p)$ to be encoded, let D_i be the set of domain values of variable X_i and let $\vec{x}_i$, with $|\vec{x}_i| = m_i$, be the Boolean variable vector that encodes X_i. Let multiset E_i consist of integers $|P^{m_i}_h|$ for $h = 0, \ldots, m_i$, and multiset T_i consist of the size of every equivalence class induced by the equivalence relation $a \equiv b$ if and only if $F|_{X_i=a} = F|_{X_i=b}$ for $a, b \in D_i$. There exists a Boolean function vector $\vec{f}$ encoding F that is fully symmetric with respect to X_i if and only if the subset-sum constraint problem of E_i summing to T_i is satisfiable under $\bowtie$ set to $\geq$.*

PROOF. ($\Longleftarrow$) Let $\vec{x_i}$ be the Boolean variable vector that encodes X_i in $\vec{f}$. The satisfiability of the subset-sum constraint of E_i summing to T_i infers that for each $t \in T_i$ there exists a multiset $A \sqsubseteq E_i$, such that $\sum(A) \geq t$. For such a multiset A, there exists a corresponding pattern set $P_A = \bigcup_j P^{m_i}_j$ for some j's with $|P_A| = \sum(A)$; for such a t, there exists a corresponding equivalence class $D \subseteq D_i$ with $t = |D|$. Since $\sum(A) \geq t$, we can always find an encoding $\mathcal{E}$ that associates every $d \in D$ to a distinct pattern in P_A. Furthermore, we can always make the cofactored function vector $\vec{f}|_{\vec{x}_i=p_a}$ of every pattern $p_a \in P_A$ unused by $\mathcal{E}$ equal to the cofactored function vector $\vec{f}|_{\vec{x}_i=p'_a}$ for any pattern $p'_a \in P_A$ used by $\mathcal{E}$. This process makes the functionality of $\vec{f}(\vec{x_1}, \ldots, \vec{x_i}, \ldots, \vec{x_p})$ unchanged under any permutation among $\vec{x_i}$. Hence the resulting $\vec{f}$ is fully symmetric with respect to X_i.

($\Longrightarrow$) Assume that there exists an encoding $\mathcal{E}$ transforming F into $\vec{f}$, which is fully symmetric with respect to X_i. Let multiset T' consist of the size of every equivalence class induced by the equivalence relation $a \equiv b$ if and only if $\vec{f}|_{\vec{x}_i=a} = \vec{f}|_{\vec{x}_i=b}$ for $a, b \in \mathbb{B}^{m_i}$. For every $t' \in T'$, there is a multiset $A \sqsubseteq E_i$ such that $\sum(A) = t'$. Also, $\bigsqcup A \sqsubseteq E_i$. Otherwise, $\vec{f}$ is not fully symmetric with respect to X_i and contradicts to the assumption. Since $\mathcal{E}$ is injective, for every $t \in T_i$ there exist $t' \in T'$ such that $t' \geq t$ and $\sum(A) \geq t$. Thus the subset-sum constraint of E_i summing to T_i is satisfiable. ∎

(So E_i consists of the binomial coefficients $\binom{m_i}{k}$ for $k = 0 \ldots m_i$.) In fact, the proof shows a constructive way to find the encoding $\mathcal{E}$ from a satisfying solution to the subset-sum constraint. Essentially the solution guarantees an injective mapping of each MV symbol to some unique code pattern. In addition, it guarantees that we can make all the patterns in P^m_h for any h have the same cofactored function vector, which in turn makes $\vec{f}$ invariant under bitwise permutation among the encoded variables of X_i.

EXAMPLE 2. *Consider the function of Example 1. Let $m_1 = 2$ for X_1. Then we have $E_1 = \{1, 2, 1\}$ since $|\{00\}| = 1$, $|\{01, 10\}| = 2$, and $|\{11\}| = 1$. Moreover, $T_1 = \{1, 1, 1, 1\}$ since $F|_{X_1=0}, F|_{X_1=1}, F|_{X_1=2}$, and $F|_{X_1=3}$ are all distinct functions. Because the subset-sum constraint of summing E_1 to T_1 is unsatisfiable, there exists no encoding of using two bits for X_1 that would make the encoded function vector fully symmetric in the two bits of variables.*

As suggested by Example 2, full symmetry may not be achieved in encoding an MV input variable of an MV function with respect to a specified code length. In reality, it is often desirable to keep the code length short. Therefore, one may search a proper code length starting from the minimum $m_{min} = \lceil \log_2 |D_i| \rceil$. If the corresponding subset-sum constraint is not satisfiable, the code length can be increased to check if symmetry encoding is possible at an expanded length. One might ask whether full symmetry is always achievable by code length expansion, and moreover whether there exists an upper bound for code length expansion. The answers to these questions are affirmative as the following theorem asserts.

THEOREM 3. *Given an MV function $F(X_1, \ldots, X_p)$ with D_i being the set of domain values of variable X_i, let multiset E_i consist of integers $|P^{m_i}_h|$ for $h = 0, \ldots, m_i$, and multiset T_i consist of the size of every equivalence class induced by the equivalence relation $a \equiv b$ if and only if $F|_{X_i=a} = F|_{X_i=b}$ for $a, b \in D_i$. There always exists a coding length m_i for X_i such that the subset-sum constraint of E_i summing to T_i is satisfiable.*

PROOF. Let $m_i = \max(t_{\max}, |T_i|+1)$, where $t_{\max}$ is largest element in T_i. Every element of the multiset $\{\binom{m_i}{k} \mid k = 1, \ldots, m_i - 1\}$ is greater than or equal to any $t \in T_i$. Moreover, the size of such a multiset is greater than or equal to $|T_i|$. It guarantees that the subset-sum constraint of E_i (corresponding to such a large m_i) summing to T_i is satisfied. ∎

EXAMPLE 3. *Continuing Example 2, let $m_1 = 3$. The corresponding E_1 becomes $\{1, 3, 3, 1\}$ since $|\{000\}| = 1$, $|\{001,$*

$010, 100\}| = 3$, $|\{011, 101, 110\}| = 3$, and $|\{111\}| = 1$, whereas T_1 remains the same. Letting $E^1 = \{1\}, E^2 = \{1\}, E^3 = \{1\}, E^4 = \{1\}$ and $t_1 = t_2 = t_3 = t_4 = 1$, apparently $\sum(E^i) \geq t_i$ for $i = 1, \ldots, 4$. The subset-sum constraint is satisfied.

With this satisfiability, we may encode D_1 domain value 0 to 000, 1 to 001, 2 to 011, and 3 to 111. Further, for the unused codes 010, 100, 101, and 110, their cofactored function vectors should be made equal to their used counterparts with the same hamming weights. That is, $\vec{f}|_{\vec{x}_1=010}$ and $\vec{f}|_{\vec{x}_1=100}$ should be made equal to $\vec{f}|_{\vec{x}_1=001}$; $\vec{f}|_{\vec{x}_1=101}$ and $\vec{f}|_{\vec{x}_1=110}$ equal to $\vec{f}|_{\vec{x}_1=011}$. So the encoded function vector can be fully symmetric to the three Boolean variables $\vec{x}_1$ of X_1.

3.3 Encoding for Partial Symmetry

Although encoding for full symmetry is guaranteed to exist, it may require impractically long code lengths. Here we explore how to find maximal partial symmetry under a code length limitation. Before presenting the main results, we define the following notation.

DEFINITION 7. *Let $p_1 \bullet p_2$ be a pattern resulted from a concatenation of patterns p_1 and p_2. For a pattern p and a pattern set P, let $p \bullet P$ be $\{p \bullet p' \mid p' \in P\}$.*

For example, for $p = 11$ and $P = \{001, 010, 101\}$, $p \bullet P = \{11001, 11010, 11101\}$ and $|p \bullet P| = |P| = 3$.

The following theorem characterizes the condition of the existence of partial symmetry.

THEOREM 4. *Given an MV function $F(X_1, \ldots, X_p)$ to be encoded, let D_i be the set of domain values of variable X_i and let $\vec{x}_i$, with $|\vec{x}_i| = m_i$, be the Boolean variable vector that encodes X_i. Let multiset $E_i(r)$, with respect to r for $0 \leq r \leq m_i - 2$, consist of integers $|p \bullet P_h^{m_i - r}|$ for all patterns $p \in \mathbb{B}^r$ and $h = 0 \ldots m_i - r$, and multiset T_i consist of the size of every equivalence class of D_i under the equivalence relation $a \equiv b$ if and only if $F|_{X_i=a} = F|_{X_i=b}$ for $a, b \in D_i$. There exists a Boolean function vector $\vec{f}$ encoding F where any two of the last $m_i - r$ variables of $\vec{x}$ form a symmetric pair if and only if the subset-sum constraint problem of $E_i(r)$ summing to T_i is satisfiable under $\bowtie$ set to $\geq$.*

PROOF. ($\Longleftarrow$) The satisfiability of the subset-sum constraint of $E_i(r)$ summing to T_i infers that for each $t \in T_i$ there exists a multiset $A \sqsubseteq E_i(r)$, such that $\sum(A) \geq t$. For such a multiset A, there exists a corresponding pattern set $P_A = \bigcup_k p_k \bullet \bigcup_{j \in J_k} P_j^{m_i - r}$ for some $p_k \in \mathbb{B}^r$ and corresponding integer set J_k with $|P_A| = \sum(A)$; for such a t, there exists a corresponding equivalence class $D \subseteq D_i$ with $t = |D|$. Since $\sum(A) \geq t$, we can always find an encoding $\mathcal{E}$ that associates every $d \in D$ to a distinct pattern in P_A. Furthermore, we can always make the cofactored function vector $\vec{f}|_{\vec{x}_i = p_a}$ of every pattern $p_a \in P_A$ unused by $\mathcal{E}$ equal to the cofactored function vector $\vec{f}|_{\vec{x}_i = p'_a}$ for any pattern $p'_a \in P_A$ used by $\mathcal{E}$. This process makes the functionality of $\vec{f}(\vec{x_1}, \ldots, \vec{x_i}, \ldots, \vec{x_p})$ unchanged under any permutation among the last $m_i - r$ variables of $\vec{x_i}$. Hence any two of the last $m_i - r$ variables of $\vec{x}$ form a symmetric pair of the resulting $\vec{f}$.

($\Longrightarrow$) Assume that there exists an encoding $\mathcal{E}$ transforming F into $\vec{f}$, such that any two of the last $m_i - r$ variables of $\vec{x}$ form a symmetric pair of $\vec{f}$. Let multiset T' consist of the size of every equivalence class induced by the equivalence relation $a \equiv b$ if and only if $\vec{f}|_{\vec{x}_i=a} = \vec{f}|_{\vec{x}_i=b}$ for $a, b \in \mathbb{B}^{m_i}$. For every $t' \in T'$, there is a multiset $A \sqsubseteq E_i(r)$ such that $\sum(A) = t'$. Also, $\bigsqcup A \sqsubseteq E_i(r)$. Otherwise, there exists two of the last $m_i - r$ variables that do not form a symmetric pair of $\vec{f}$ and contradicts to the assumption. Since $\mathcal{E}$ is injective, for every $t \in T_i$ there exist $t' \in T'$ such that $t' \geq t$ and $\sum(A) \geq t$. Thus the subset-sum constraint of $E_i(r)$ summing to T_i is satisfiable. ■

```
SymmetryEncoding
  input: an MV function $F(X_1, ..., X_p)$
  output: encoded Boolean function vector $\vec{f}(\vec{x}_1, ..., \vec{x}_p)$
  begin
    for $i$ from 1 to $p$
      Res[$i$] := $\emptyset$;
      $T_i$ := GenerateT($F$, $X_i$);
      num_input_bit := $\lceil \lg |D_i| \rceil$;
      num_relax_bit := 0;
      min_relax_bit := $\infty$;
      while true
        result := Solving(num_input_bit, num_relax_bit, $T_i$);
        if result = FAIL
          if SYMM-MAXS
            num_input_bit := num_input_bit +1;
          else // SYMM-MINL or SYMM-TRAD
            num_relax_bit := num_relax_bit +1;
        else
          if SYMM-MAXS or SYMM-MINL
            Res[$i$] := result;
            break;
          else // SYMM-TRAD
            if min_relax_bit > num_relax_bit
              min_relax_bit := num_relax_bit;
              num_input_bit := num_input_bit +1;
              tmp_result := result;
            else
              result := tmp_result;
              Res[$i$] := result;
              break;
    $\vec{f}$ := Encode($F$, Res);
    return $\vec{f}$;
  end

Solving(num_input_bit, num_relax_bit, T)
  begin
    num_copy := $2^{(\text{num\_relax\_bit})}$;
    level := num_input_bit − min_relax_bit;
    $E$ := $\emptyset$;
    for $j$ from 1 to num_copy
      $E$ := $E \sqcup$ BinomialCoefficients(level);
    return SolveSubsetSum($E$,$T$);
  end
```

Figure 1: Algorithm: Symmetry encoding

The above parameter r is referred to as the *number of relaxed bits* for partial symmetry encoding. Notice that Theorem 2 is a special case of Theorem 4 with r setting to 0.

The multiset $E_i(r)$ can be intuitively understood as the multiset union of 2^r copies of the binomial coefficients $\binom{m_i - r}{k}$ for $k = 0, \ldots, m_i - r$. For example, given encoding length $m_i = 4$ and we relax symmetry with 2 bits, $E_i(2) = \{1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1\}$ which is 2^2 copies of binomial coefficients $\binom{4-2}{k}$ for $k = 0, 1, 2$.

3.4 Symmetry Encoding Algorithm

Figure 1 shows the procedure for symmetric encoding with three optional strategies, including SYMM-MAXS, which pursues maximal symmetry without code length limitation, SYMM-MINL, which pursues maximal degree of symmetry with respect to the minimum code length limitation, and SYMM-TRAD, which admits code length increase until the number of relaxed bits stops decreasing.

In Figure 1, for every input variable X_i in the given MV function F, we first calculate the target multiset T_i by Theorem 2, followed by searching an encoding that makes the encoded function $\vec{f}$ symmetric with respect to X_i under the required strategy. In each iteration of the while loop, if the subset-sum constraint is proved to be satisfiable, the process

```
SolveSubsetSum(E, T)
  begin
  01  table := BuildSubsetSumTable(E, largest element in T);
  02  F := CheckFalse(table,|E|,T);
  03  if F ≠ ∅
  04      return FAIL;
  05  return Search(table, E, T);
  end

CheckFalse(table, row_index, T)
  begin
  01  false_position := (∅, constant zero);
  02  for t ∈ T
  03      if table[row_index][t] = FALSE
  04          false_position := false_position ⊔{t};
  05  return false_position;
  end
```

Figure 2: Algorithm: Subset-sum constraint solving

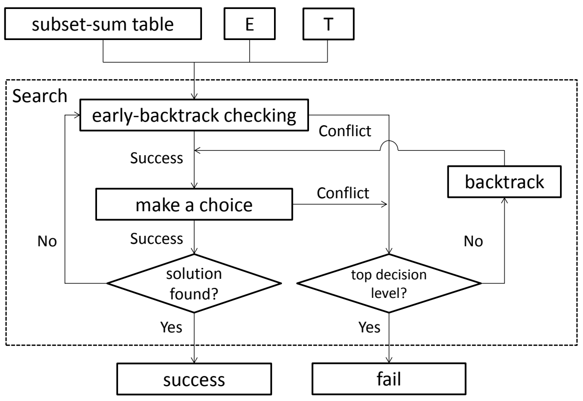

Figure 3: Search flow for subset-sum constraint solving

proceeds to the next X_i. If not, for the next iteration, to pursue full symmetry with expanded coding length (respectively partial symmetry with minimum coding length), increase the coding length "num_input_bit" (respectively the number of relaxed bit "num_relax_bit") by one. If both coding length expansion and bit relaxation are required (in the case of SYMM-TRAD), we search a coding length with a relatively high degree of symmetry in a heuristic way. For every current minimum number of relaxation bits "min_relax_bit", we try to increase encoding length and intend to find a smaller number of relaxation variables. If this attempt fails, there is no such coding length, and we conclude that the coding length resulting in the current minimum number of relaxation bits is an appropriate length.

4. SUBSET-SUM CONSTRAINT SOLVING

This section presents a decision procedure, outlined in Figure 2, of subset-sum constraint solving for symmetry encoding. Given two multisets E and T as specified in Definition 5, the procedure *SolveSubsetSum* first builds a table for effective solution search using dynamic programming (based on the standard approach to the conventional subset-sum problem [6]). It then performs *CheckFalse* for fast screening by solving a single-target subset-sum problem with respect to each $t \in T$. If there is a target cannot be achieved, there is no solution to the original problem, and the procedure returns failure. Otherwise, it invokes *Search* to find a solution. The major steps are explained in detail as follows.

4.1 Building Subset-sum Table

Let the multisets $E = \{e_1, \ldots, e_m\}$ and $T = \{t_1, \ldots, t_k$ be sorted in an ascending order. *BuildSubsetSumTable* constructs a two-dimensional table, whose rows and columns are indexed by the sequence $0, e_1, e_2, \ldots, e_m$ and sequence $-e_m + 1, -e_m + 2, \ldots, t_k$, respectively. The entry at the i^{th} row and the column indexed by j, denoted $S[i, j]$, corresponds to a predicate indicating whether the values chosen from the first i items in E can sum to a value greater than or equal to j.

For $j \leq 0$, $S[i, j]$ is `True`. For $i = 0$ and $j > 0$, $S[i, j]$ is `False`. The other entries should satisfy

$$S[i, j] = S[i - 1, j - e_i] \vee S[i - 1, j], \tag{1}$$

where the truth of $S[i, j]$ is determined by two possibilities depending on whether e_i is chosen or not. If yes, $S[i-1, j-e_i]$ is `True`; if no, $S[i - 1, j]$ is `True`.

Example 4. *Continue Example 3. The corresponding subset-sum table can be found in Figure 4.*

4.2 Solution Search for Subset-sum Constraints

For $|T| = 1$, the solution can be spotted from the table directly. We simply check whether the table entry at the intersection of the $|E|^{\text{th}}$ row and the column indexed by $t \in T$ is `True`. If yes, we may trace back for finding the corresponding sub-multiset of E. Otherwise, no solution exists. However, to solve a constraint with multiple summation targets, we have to search in the table to find a solution that satisfies all summation targets simultaneously.

As shown in Figure 3, the subset-sum constraint solving procedure performs a series of decisions that determine at each row whether or not the corresponding row element should be used, and if yes, which entry $t \in T$ should use the row element. Start from the $|E|^{\text{th}}$ row of the table, it recursively goes back and forth to visit each row to make the decisions. When a decision is made, we modify the corresponding t by subtracting the row element from it. When a conflict occurs due to a decision, backtrack and make another decision. The search procedure terminates either when all $t \in T$ are non-positive after a decision, which means the constraint is satisfiable, or no more decision can backtrack, which means the constraint is unsatisfiable.

Let the integer multiset $T(i)$ list the current status of each $t \in T$ in the visit to the i^{th} row and $T(|E|) = T$ as the initial condition. Make sure the values of table entries corresponding to $t' \in T(i)$ are all `True` during search. Otherwise, there exists $t' \in T(i)$ that cannot be fulfilled by the elements chosen from the first i elements of E. During the visit of the i^{th} row, we copy $T(i)$ to $T(i - 1)$, followed by choosing and modifying a $t' \in T(i - 1)$ with the row element e_i. The candidates of the chosen t' depend on how we visit the row. If the visit takes place at the $|E|^{\text{th}}$ row for the first time or follows the visit of the $(i + 1)^{\text{st}}$ row, apply the function *CheckFalse*(table,$i - 1$,$T(i - 1)$) to check the size of `false_position`, a list of $t' \in T(i - 1)$ whose corresponding table entry is false and needs to be modified. If the size is larger than one, a conflict occurs and a backtrack to the $(i + 1)^{\text{st}}$ row is needed, since more than one t' needs to be modified by e_i simultaneously. If the size equals one, modify the only t' in `false_position` with the row element e_i. If the size equals zero, we can either choose and modify exactly one $t' \in T(i - 1)$ with e_i and record the others as the candidates after backtracking from this decision, or just leave e_i unused. We heuristically choose a t' first, and leave it unused when the previous decision encounters a conflict. On the other hand, if the visit of the i^{th} row backtracks from the visit of $(i - 1)^{\text{st}}$ row, we choose a t' from the recorded candidates. If all choices are disproved for a decision, i.e., no candidate remains and the choice of leaving the row element unused is disproved, we backtrack again.

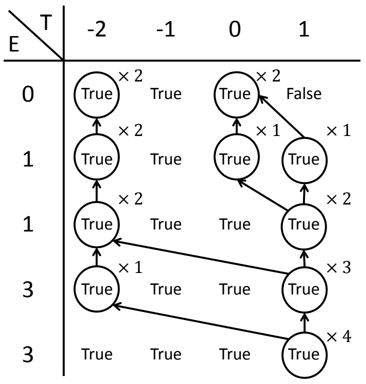

Figure 4: Search traces

Example 5. *Continuing Example 4, the search traces are shown in Figure 4. The table entries marked with circles are the current statuses $t' \in T(i)$ in the i^{th} row. The integer labelled beside a circle signifies how many times the corresponding column element present in $T(i)$. First, search starts from Row 4 and $T(4) = \{1, 1, 1, 1\}$. (So the number 4 beside the corresponding circle signifies the column element 1 appears four times in $T(4)$.) We choose one of the entries in $T(4)$, say the first entry, 1, to use the row element $e_4 = 3$ and propagate to Row 3 with the new multiset $T(3) = \{-2, 1, 1, 1\}$ for the first entry is updated by $1 - 3 = -2$. Repeat the process, finally all t's in $T(0) = \{-2, -2, 0, 0\}$ are non-positive in Row 0. Search is successful and the constraint is satisfied.*

5. EXPERIMENTAL RESULTS

We implemented the proposed symmetry encoding algorithm in Python. The experiments were conducted on a Linux machine with a Xeon 2.53GHz CPU and 48GB RAM. We compare our symmetry encoding with other methods, including NAIVE, ONE-HOT, ESPRESSO [4], and MVSIS [10] encodings. We further applied three symmetry encoding strategies for comparison, including 1) maximizing symmetry degree without code length restriction, denoted SYMM-MAXS, 2) trading between symmetry degree and code length, denoted SYMM-TRAD, and 3) maximizing symmetry degree under a minimum code length, denoted SYMM-MINL.

The finite state machine examples of the LGSynth89 benchmark suite were taken to study state encoding. In these benchmarks, only the state variable is multi-valued. Table 1 shows the results of state encoding by six methods, namely, SYMM-TRAD (Columns 5-7), SYMM-MINL (Columns 8-10), NAIVE (Columns 11 and 12), ONE-HOT (Columns 13 and 14), ESPRESSO (Columns 13 and 14), and MVSIS (Columns 15 and 16). Columns 2, 3, and 4 list the number of input bits, number of output bits, and number of states of the benchmarks, respectively. For the statistics of each encoding method, the columns denoted "sb," "ratio," and "time" list the encoding length for the state variable, the ratio in percentage of the number of symmetry pairs to the number of input pairs of the entire circuit after encoding, and the CPU time in seconds for symmetry encoding, respectively. For ONE-HOT and ESPRESSO, the encoding length equals the number of states; for MVSIS, the encoding length is minimum. As can been seen, symmetry encoding, especially SYMM-TRAD, achieved a high degree of symmetry compared to other methods. Comparing SYMM-TRAD and SYMM-MINL, we found that expanding code length by one may effectively increase the degree of symmetry. Note that, for circuit `planet`, MVSIS is not applicable due to its representation limitation of up to 32 values for an MV variable.

To further study the encoding quality, the encoded finite state machines were further optimized by ABC [2] using script `st; dsd; st; dc2; dc2; dc2; dc2; dc2`. The results are shown in Table 2. For each encoding method, the number of and-inverter graph (AIG) nodes, denoted "#and," and the number of logic levels, denoted "#lev," are shown. The data reveal that in most cases MVSIS encoding achieved the best synthesis quality in terms of circuit sizes. It is not surprising since MVSIS encoding targets logic minimization. On the other hand, it can be seen that SYMM-MINL achieved reasonable circuit quality (comparable to NAIVE, and superior to ONE-HOT and ESPRESSO in most cases).

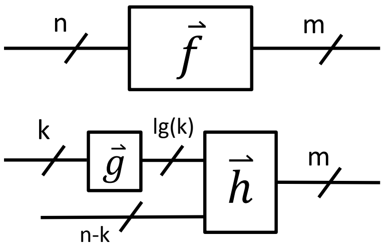

Figure 5: Functional decomposition in the presence of symmetry.

In Tables 1 and 2, we did not show the results of SYMM-MAXS due to its impractical circuit sizes. Nevertheless, as mentioned in Section 1, symmetry can be exploited for functional decomposition. We studied how the decomposition architecture of Figure 5 can be exploited to minimize the (highly symmetric but huge) circuits produced by SYMM-MAXS. As shown in the figure, an original function vector $\vec{f}$ is decomposed into a composition of vector $\vec{g}$ and vector $\vec{h}$, where $\vec{g}$ is a circuit counting the hamming weight of its inputs. The synthesized circuit of $\vec{f}$ is compared with the composite circuit of separately synthesized $\vec{g}$ and $\vec{h}$, all under ABC script `st; dc2; dc2; dc2; dc2; dc2`. The results are shown in Table 3, where Column 2 shows the code length for state encoding, Column 3 shows the ratio in percentage of the number of symmetry pairs to the number of input pairs of the entire circuit after encoding, Column 4 shows the CPU time for SYMM-MAXS encoding, Columns 5 and 6 show the numbers of AIG nodes and logic levels, respectively, after decomposition-based synthesis, and Column 7 (Column 8) shows the ratio of the number of reduced AIG nodes (increased logic levels) to the number of nodes (levels) of the circuit before decomposition. As can be seen, functional decomposition under the presence of symmetry can achieve effective circuit size reduction with relatively small circuit level increase. (Note that the ratios shown in Column 2 are not 100% because the symmetric pairs were counted over all input pairs, not just the state variable pairs.)

To study the scalability and quality of our symmetry encoding algorithm, we randomly generated 100 (single-input and single-output) MV functions, which all have fixed four output values and have 2^k input values for $k = 5, \dots, 14$ (ten functions for each k were generated). For our symmetry encoding, most of the cases are solved within 10 seconds; the most time-consuming cases (SYMM-TRAD on the ten functions with 16384 input values) took 73.88 seconds on average. The results suggested that our encoding algorithm can handle MV functions with tens of thousands of input values in reasonable

Table 1: State Encoding: Degree of Symmetry

Name	#in	#out	#state	SYMM-TRAD			SYMM-MINL			NAIVE		ONE-HOT / ESPRESSO		MVSIS	
				sb	ratio	time	sb	ratio	time	sb	ratio	sb	ratio	sb	ratio
bbara	4	2	10	5	19.44	0.12	4	7.14	0.06	4	3.57	10	1.1	4	3.57
bbsse	7	7	16	5	4.55	1.63	4	0	0.75	4	0	16	1.19	4	0
bbtas	2	2	6	3	10.00	0.00	3	10.00	0.00	3	0	6	0	3	0
beecount	3	4	7	4	14.29	0.03	3	0	0.01	3	0	7	0	3	0
cse	7	7	16	5	6.06	1.71	4	1.82	0.77	4	0	16	0	4	0
dk14	3	5	7	4	14.29	0.03	3	0	0.01	3	0	7	0	3	0
dk15	3	5	4	3	20.00	0.01	2	0	0.01	2	0	4	0	2	0
dk16	2	3	27	6	10.71	0.07	5	0	0.02	5	0	27	0	5	0
dk17	2	3	8	4	20.00	0.01	3	0	0.01	3	0	8	0	3	0
dk27	1	2	7	4	30.00	0.01	3	0	0.00	3	0	7	0	3	0
dk512	1	3	15	5	20.00	0.03	4	0	0.01	4	0	15	0	4	0
donfile	2	1	24	5	4.76	0.03	5	4.76	0.02	5	0	24	0	5	0
ex1	9	19	20	6	5.71	177.88	5	1.10	26.70	5	0	20	0	5	0
ex2	2	2	19	6	21.43	0.04	5	4.76	0.02	5	0	19	0	5	0
ex3	2	2	10	5	28.57	0.02	4	6.67	0.01	4	0	10	0	4	0
ex4	6	9	14	5	5.45	0.59	4	0	0.43	4	0	14	0	4	0
ex5	2	2	9	5	28.57	0.02	4	6.67	0.01	4	0	9	0	4	0
ex6	5	8	8	4	8.33	0.12	3	0	0.08	3	0	8	0	3	0
ex7	2	2	10	5	28.57	0.03	4	6.67	0.01	4	0	10	0	4	0
keyb	7	2	19	6	7.69	2.99	5	1.52	1.19	5	0	19	0	5	0
lion	2	1	4	3	28.57	0.00	2	6.67	0.00	2	0	4	0	2	0
lion9	2	1	9	5	30.00	0.02	4	0	0.01	4	0	9	0	4	0
mc	3	5	4	3	20.00	0.01	2	0	0.01	2	0	4	0	2	0
modulo12	1	1	12	4	10.00	0.01	4	10.00	0.00	4	0	12	0	4	0
planet	7	19	48	6	1.28	9.65	6	1.28	9.64	6	0	48	0	-	-
s1	8	6	20	6	6.59	13.89	5	1.28	4.12	5	0	20	0	5	0
s8	4	1	5	4	21.43	0.03	3	4.76	0.01	3	0	5	0	3	0
shiftreg	1	1	8	4	30.00	0.01	3	0	0.00	3	0	8	0	3	0
sse	7	7	16	5	4.55	1.59	4	0	0.73	4	0	16	1.19	4	0
styr	9	10	30	6	2.86	80.28	5	0	21.38	5	0	30	0	5	0
tav	4	4	4	3	14.29	0.02	2	0	0.01	2	0	4	0	2	0
tbk	6	3	32	6	4.55	1.47	5	0	0.72	5	0	32	0	5	0
train11	2	1	11	5	28.57	0.03	4	6.67	0.01	4	0	11	0	4	0
train4	2	1	4	3	40.00	0.01	2	16.67	0.00	2	16.67	4	6.67	2	16.67

Table 2: State Encoding: Circuit Synthesis

Name	SYMM-TRAD		SYMM-MINL		NAIVE		ONE-HOT		ESPRESSO		MVSIS	
	#and	#lev	#and	#lev	#and	#lev	#and	#lev	#and	#lev	#and	#lev
bbara	125	11	80	8	65	7	147	12	121	10	**60**	7
bbsse	188	12	121	9	119	8	243	17	191	11	**118**	9
bbtas	**19**	6	**19**	6	23	4	47	6	41	5	21	6
beecount	40	8	**32**	6	59	6	154	11	123	10	57	7
cse	332	15	211	12	211	12	347	19	321	15	**184**	10
dk14	110	9	99	7	85	7	169	11	147	9	**81**	8
dk15	87	8	57	7	54	6	105	9	101	9	**50**	6
dk16	480	12	288	12	287	11	488	15	453	14	**265**	11
dk17	94	8	58	7	56	6	126	10	104	10	**45**	6
dk27	44	8	28	5	22	4	48	6	43	5	**18**	5
dk512	121	8	63	7	**59**	7	134	9	115	7	64	8
donfile	196	10	196	10	224	9	426	23	396	23	**97**	10
ex1	636	14	460	13	374	11	700	21	531	18	**219**	13
ex2	408	13	180	10	169	10	323	17	248	11	**165**	10
ex3	155	10	74	7	**72**	7	136	11	120	8	80	7
ex4	160	9	80	8	92	7	177	14	148	9	**74**	6
ex5	104	10	**72**	7	**72**	8	136	13	103	8	76	8
ex6	151	10	130	7	**102**	8	227	15	176	9	115	7
ex7	77	9	78	8	**70**	7	137	11	128	10	81	7
keyb	518	16	290	15	326	16	378	24	330	25	**262**	15
lion9	99	10	56	8	69	8	116	13	70	8	**55**	5
lion	16	6	**9**	4	10	4	37	8	39	7	13	8
mc	33	6	17	5	16	4	43	7	42	7	**13**	4
modulo12	34	8	34	8	35	6	94	6	91	6	**16**	5
planet	**575**	12	**575**	12	613	11	1711	41	1657	41	-	-
s1	849	16	458	13	460	16	596	25	553	21	**316**	15
s8	85	12	46	9	**42**	7	83	8	68	8	48	7
shiftreg	23	8	24	5	14	4	44	5	34	4	**0**	0
sse	188	12	162	9	**107**	9	221	17	191	11	118	9
styr	1022	17	575	14	644	16	942	26	927	27	**446**	14
tav	21	5	**14**	4	16	4	27	10	32	7	16	4
tbk	1499	19	911	15	887	16	1558	37	1109	29	**210**	15
train11	77	10	**62**	8	83	9	139	13	94	14	76	9
train4	17	5	10	4	13	4	31	6	25	5	**9**	5

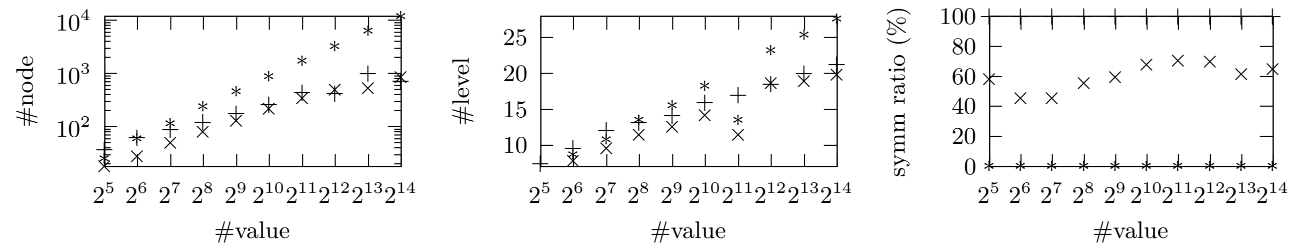

Figure 6: Scalability study. Results of SYMM-TRAD, SYMM-MINL, and NAIVE are plotted in "+," "×," and "∗," respectively.

Table 3: Circuit Reduction by Functional Decomposition

Name	sb	ratio	time	#and	#lev	size(−)	lev(+)
bbara	9	47.44	1.63	463	23	0.66	0.35
bbtas	5	47.62	0.01	70	13	0.23	0.44
beecount	6	41.67	0.08	132	17	0.08	0.70
dk14	6	41.67	0.05	197	16	0.57	0.14
dk15	3	20.00	0.01	68	10	0.22	0.25
dk17	7	58.33	0.06	164	17	0.66	0.21
dk27	6	71.43	0.01	75	12	0.28	0.09
dk512	14	86.67	5.92	2169	27	0.85	0.13
ex3	9	65.45	0.23	391	22	0.74	0.29
ex5	8	62.22	0.11	279	20	0.66	0.33
ex6	7	31.82	1.21	292	19	0.71	0.27
ex7	9	65.45	0.22	353	22	0.73	0.22
lion	3	62.22	0.00	231	19	0.68	0.27
lion9	8	30.00	0.09	15	5	0.06	-0.17
mc	3	20.00	0.01	28	7	0.15	0.17
modulo12	11	83.33	0.40	582	22	0.83	0.00
s8	4	21.43	0.03	92	12	-0.08	0.00
shiftreg	7	75.00	0.01	120	15	0.15	0.36
tav	3	14.29	0.02	21	5	0.00	0.00
train11	9	65.45	0.22	315	21	0.60	0.11
train4	3	40.00	0.00	16	6	0.06	0.20

runtime.

As for encoding quality, three encoding methods SYMM-TRAD, SYMM-MINL, and NAIVE, were compared in Figure 6, where each dot corresponds to the average value of ten encoded circuits. The encoded circuits were simplified using the same ABC synthesis script as that of Table 2. As can be seen, SYMM-TRAD and SYMM-MINL are superior to NAIVE in terms of AIG sizes, logic levels, and symmetry degrees. Moreover, the quality gap tends to increase for large test cases. (We did not compare with MVSIS due to its limitation of handling up to 32 MV values, not compare with ONE-HOT and ESPRESSO due to their inferior qualities suggested in Table 2.)

6. CONCLUSION AND FUTURE WORK

This paper formulated the Symmetry Encoding Problem in terms of a system of subset-sum constraints. We showed that code length and symmetry degree can be adjusted for a tradeoff. A decision procedure was proposed for effective subset-sum constraint solving. Experiments suggested that symmetry encoding is advantageous in terms of increasing symmetry degree while circuit size and logic depth can be maintained comparable with other encoding methods.

As this paper focused on independent MV input encoding, for future work, we would like to consider dependent MV input encoding. Moreover, encoding taking into account symmetric pairs for variables belonging to two different MV inputs could be considered. One might relax the injectivity restriction of our encoding to allow one-to-many mappings. As our method opens up a new direction, encoding for other functional properties, such as unateness, can be considered. Also it would be interesting to identify other applications of our decision procedure of subset-sum constraint solving.

REFERENCES

[1] P. Ashar, S. Devadas, and R. Newton. *Sequential Logic Synthesis*, Kluwer Academic Publishers, 1992.

[2] Berkeley Logic Synthesis and Verification Group. *ABC: A system for sequential synthesis and verification.* `http://www.eecs.berkeley.edu/~alanmi/abc/`

[3] R. K. Brayton and S. P. Khatri. Multi-Valued Logic Synthesis. *Proc. Int'l Conf. on VLSI Design*, pp. 196-205, 1999.

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

[5] C.-W Chang, C.-K, Cheng, P. Suaris, M. Marek-Sadowska. Fast Post-placement Rewiring Using Easily Detectable Functional Symmetries. In *Proc. Design Automation Conference*, pp. 286-289, 2000.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, MIT Press, 2001.

[7] K.-H. Chang, I. L. Markov, V. Bertacco. Post-placement Rewiring and Rebuffering by Exhaustive Search for Functional Symmetries. In *Proc. Int'l Conf. on Computer-Aided Design*, pp. 56-63, 2005.

[8] G. De Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli. Optimal State Assignment for Finite State Machines. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 4(3): 269-285, 1985.

[9] G. De Micheli. *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.

[10] M. Gao, J.-H. Jiang, Y. Jiang, Y. Li, S. Sinha, and R. K. Brayton. MVSIS. In *Proc. Int'l Workshop on Logic Synthesis*, 2001.

[11] Y.-J. Jiang and R. K. Brayton. Don't Cares and Multi-Valued Logic Network Minimization. In *Proc. Int'l Conf. on Computer-Aided Design*, pp. 520-525, 2000.

[12] J.-H. Jiang, Y.-J. Jiang, and R. K. Brayton. An Implicit Method for Multi-Valued Network Encoding. In *Proc. Int'l Workshop on Logic Synthesis*, 2001.

[13] J.-H. Jiang, A. Mishchenko, and R. K. Brayton. Reducing Multi-Valued Algebraic Operations to Binary. In *Proc. Design, Automation, and Test in Europe*, pp. 10752-10757, 2003.

[14] M.-Y. Li. *Pseudo-Boolean Constraint Formulation of Symmetry Boolean Encoding for Multi-Valued Functions.* Master Thesis, National Taiwan Univeristy, 2011.

[15] L. Lavagno, S. Malik, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. MIS-MV: Optimization of Multi-level Logic with Multiple-valued Inputs. In *Proc. Int'l Conf. on Computer-Aided Design*, pp. 560-563, 1990.

[16] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli. Optimum Functional Decomposition Using Encoding. *Proc. Design Automation Conference*, pp. 408-414, 1994.

[17] A. Mishchenko. Fast Computation of Symmetries in Boolean Functions. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 22(11):1588-1593, Nov. 2003.

[18] C. Scholl, D. Moller, P. Molitor, and R. Drechsler. BDD Minimization Using Symmetries. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 18(2): 81-100, Feb. 1999.

[19] T. Villa and A. L. Sangiovanni-Vincentelli. NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 9(9): 905-924, Sep. 1990.

[20] K.-H. Wang, C.-M. Chan, and J.-C. Liu. Simulation and SAT-based Boolean Matching for Large Boolean Networks. In *Proc. Design Automation Conference*, pp. 396-401, 2009.