# Reducing Multi-Valued Algebraic Operations to Binary*

Jie-Hong R. Jiang
Dept. of EECS
Univ. of California, Berkeley
jiejiang@eecs.berkeley.edu

Alan Mishchenko
Dept. of ECE
Portland State University
alanmi@ece.pdx.edu

Robert K. Brayton
Dept. of EECS
Univ. of California, Berkeley
brayton@eecs.berkeley.edu

## ABSTRACT

*Algebraic operations were developed for binary logic synthesis and extended later to apply to multi-valued (MV) logic. Operations in the MV domain were considered more complex and slower. This paper shows that MV algebraic operations are essentially as easy as binary ones, with only a slight overhead (linear in the size of the expression) in transformation into and out of the binary domain. By introducing* co-singleton *sets as a new basis, any MV sum-of-products expression can be rewritten and passed to a binary logic synthesizer for fast execution; the optimized results can be directly interpreted in the MV domain. This process, called EBD, reduces MV algebraic operations to binary. A pure MV operation differs mainly from its corresponding EBD one in that the former possesses "semi-algebraic" generality, which has not been implemented for the binary logic. Experiments show that the proposed methods are significantly faster, with little or no loss in quality when run in comparable scripts of sequences of logic synthesis operations.*

## 1. INTRODUCTION

In high-level design, system descriptions are inherently multi-valued, motivating the pursuit of optimality directly in the MV domain. The MVSIS project [2] is an example. To see how one can benefit from MV optimization, compare the following two design flows:

1. MV optimization → encoding → binary optimization
2. Encoding → binary optimization

Given a design instance, suppose the same binary encoding is applied in these two flows. Let the first flow have semi-algebraic optimization power in the MV domain but only algebraic power in the binary domain. This flow may yield results which could only correspond to the use of non-algebraic optimization in the second flow. This is one distinction between these two flows. Another is that the encoding in the first flow can be selected to depend on the results of the MV optimization. In this paper we are concerned with the MV optimization step.

Traditionally, logic synthesizers are designed for binary optimization. Thus, to minimize MV expressions, they must be encoded first into binary before logic synthesizers can come into play. The disadvantages are twofold: first, the optimization is restricted to a particular encoding; second, compact structures might be destroyed by the encoding, making it more difficult to detect and extract good common expressions. Moreover, since MV optimization problems were considered to be more sophisticated than binary ones; it was not obvious if optimization, directly in the MV domain, could be made efficient. In this paper we overcome all of these obstacles at once. To achieve this, without resorting to binary encoding, we rewrite MV expressions in terms of *co-singleton* sets, while preserving the structures of these expressions. Thus, the new representations, which are *binary in disguise*, can be processed by a binary logic synthesizer. In addition, the results directly reflect optimized structures in the MV domain.

This approach, called *EBD* (Execution in Binary-in-Disguise), can be significantly faster than previous MV algebraic operations. In [1, 5, 6], semi-algebraic[1] operations for multi-valued input, binary output functions have been implemented in MVSIS for synthesizing multi-valued multi-level networks. The methods are fairly complex and tend to be slow on large examples, but were a significant improvement over previously known methods [8]. In contrast, their algebraic binary counterparts are very fast. An MV algebraic method [10] has also been defined for the multi-valued *output* case. It uses the the MIN and MAX operators instead of AND and OR, and thus is not directly comparable with our formulation.

Semi-algebraic in the binary domain refers to using the identity $xx = x$. Thus it allows the following factorization (semi-algebraic):

$$abd + aef + bcdf + cef = (cf + a)(bd + ef).$$

Although semi-algebraic methods have been proposed for the binary domain, they have not been implemented because they might slow down the algebraic operations significantly. Thus, in this paper, all algebraic operations in the binary domain are restricted to purely algebraic. In the rest of this paper, if the term algebraic is used in referring to the MV domain it means the more general, semi-algebraic, and when used in the binary domain, means purely algebraic rather than semi-algebraic.

We show how all MV semi-algebraic operations can be mapped into operations on purely binary-input functions using co-singletons to rewrite each multi-valued literal. Then

*The first and third authors would like to thank the California State Micro program and our industrial sponsors, Cadence and Synplicity, as well as the sponsorship of the SRC under contract 683.004. The second author gratefully acknowledges the support of Intel.

[1]In logic synthesis it is common to refer to algebraic as those operations which manipulate expressions like polynomials. In the MV domain, because purely algebraic operations yield very limited optimality, the generalization to semi-algebraic was made to include the absorption rules.

the binary algebraic operations of *extraction*, *factoring*, *decomposition*, and *algebraic division* can be applied. After the result is obtained, the answer is translated back into the original MV domain. This set of MV operations are referred to as *the EBD operations*.

Although the binary execution of the EBD operations do not have semi-algebraic equivalents, they have no obvious loss of quality; there are results that can be obtained using a multi-valued method, which cannot be obtained using the corresponding EBD method and vice versa. However often they only differ in how the flexibility available during the algebraic division process is used in the MV domain, each result representing opposite extremes.

The rest of this paper is organized as follows. In Section 2, we define our representations for MV logic. Section 3 introduces the co-singleton transform with an EBD factorization example. For each algebraic operation, the pure MV method and its corresponding EBD method are compared in Section 4. We then extend the discussion to non-algebraic operations in Section 5. Experimental results and conclusions are given in Sections 6 and 7.

## 2. PRELIMINARIES

As a generalization of Boolean functions, we define:

DEFINITION 1. *A* multi-valued function $\mathcal{F}(a, b, \ldots)$ *is a function* $\mathcal{F} : A \times B \times \cdots \mapsto T$, *where* $a$, $b$, $\ldots$ *are* multi-valued variables *taking on values from sets* $A$, $B$, $\ldots$ *respectively, and* $T$ *is the co-domain of* $\mathcal{F}$.

In particular, we consider finite sets with size $\geq 2$. (In this paper we use natural numbers to represent elements of a set. An $n$-element set has elements $0, 1, \ldots, n-1$; no ordering among the elements is implied.) Like Boolean functions, MV functions can be expressed in sum-of-products (SOP) forms. For example,

$$\mathcal{F}^{\{\tau\}} = (a^{S_a} b^{S_b} \cdots) + (a^{S'_a} b^{S'_b} \cdots) + \cdots,$$

where $\emptyset \subset S_a, S'_a, \ldots \subseteq A$, $\emptyset \subset S_b, S'_b, \ldots \subseteq B$, $\ldots$, and $\tau \in T$.

Also we define a *co-singleton* as follows.

DEFINITION 2. *Given a set* $U$ *(with size* $\geq 2$*) and a single element set (a singleton)* $S \subset U$, *the* co-singleton *of* $S$ *is the complement set of* $S$ *with respect to* $U$.

It is immediate that:

PROPOSITION 1. *Given a set* $U$ *(with size* $\geq 2$*), any subset* $S \subset U$ *has a unique representation in the form of a conjunction of co-singletons* $\subset U$.

## 3. THE CO-SINGLETON TRANSFORM

Consider the following problem. Given an arbitrary multi-valued SOP expression $\mathcal{E}$ and an oracle $\Omega$ which optimally factors a binary SOP input, can we take advantage of $\Omega$ to factor $\mathcal{E}$ optimally? If yes, how?

To achieve this, we must have two criteria:

1. Transform $\mathcal{E}$ into $\mathcal{E}'$ which "looks like" binary, and
2. Transform the resultant output of $\Omega$, $\mathcal{E}''$, back to an expression that "directly" reflects an optimally factored form of $\mathcal{E}$.

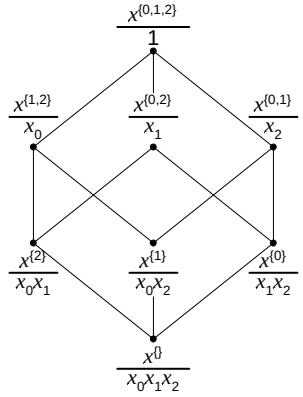


**Figure 1: The Hasse diagram of the power set of $X = \{0, 1, 2\}$: For an MV variable $x$ taking on values from $X$, all of its literals can be expressed by products of co-singletons. The new expressions are listed under the corresponding literals.**

Obviously any binary encoding, *which associates a value of an MV variable with a distinct code of a vector of binary variables*, satisfies the first criterion. However, none satisfies the second for all possible $\mathcal{E}$. For example, 1-hot codes [3], complemented 1-hot codes [4] and all logarithmic codes fail. Historically, there have been many instances of encoding multi-valued applications into binary. For example, the initial version of ESPRESSO [3] was binary. It used the idea of treating the multiple outputs of a PLA as a single multi-valued input. These were coded with a 1-hot representation, and don't cares were added to indicate that two or more values are present at the same time or all values are absent. Also, most MDD implementations [7] are simply wrappers under which MV variables are encoded with some logarithmic code and then a regular BDD package is used.

To illustrate multi-valued factoring, consider the following MV expression with two 4-valued variables $a$ and $b$,

$$a^{\{2,3\}} b^{\{0,1\}} + a^{\{0,3\}} b^{\{1,2\}} + a^{\{1,2\}} b^{\{0,3\}} + a^{\{0,1\}} b^{\{2,3\}}.$$

For $x^{S_1} x^{S_2} = x^{S_1 \cap S_2}$, it can be factored as

$$= (a^{\{0,2,3\}} b^{\{0,1,2\}} + a^{\{0,1,2\}} b^{\{0,2,3\}}) \cdot (a^{\{1,2,3\}} b^{\{0,1,3\}} + a^{\{0,1,3\}} b^{\{1,2,3\}}).$$

Observe that, unlike binary SOP expressions, multi-valued ones still have, in effect, disjunctive operations (due to MV literals) in each product term. They are a restricted form of 3-level expressions. To eliminate such disjunctive operations, we need to re-express each MV literal in a pure product form. Also we require a bijection between the new expressions and the original MV literals. More importantly, after factorization, we should be able to recover MV literals.

Here is one solution, called the *co-singleton transform*. Let $x$ be a variable taking on values from $X = \{0, \ldots, n-1\}$. In the rest of this paper, we use $x_i$ to denote $x^{\{0,\ldots,i-1,i+1,\ldots,n-1\}}$ (i.e. the literal with co-singleton set $\{0, \ldots, i-1, i+1, \ldots, n-1\}$), and use $\overline{x}_i$ to denote $x^{\{i\}}$ (i.e. the literal with singleton set $\{i\}$). Notice that $\overline{x}_i$ is equivalent to the product term $x_0 \cdots x_{i-1} x_{i+1} \cdots x_{n-1}$. From Proposition 1, we know that any literal of $x$ can be uniquely expressed as a product of some co-singleton literals. Figure 1 illustrates the case for

$n = 3$. With this convention, the previous example can be rewritten as

$$a_0a_1b_2b_3 + a_1a_2b_0b_3 + a_0a_3b_1b_2 + a_2a_3b_0b_1.$$

With no excuse for rejecting wrong formats, $\Omega$ factors this expression in the binary domain, yielding

$$= (a_1b_3 + a_3b_1)(a_0b_2 + a_2b_0).$$

One can check that in this case the (EBD) factorization gives the same result as the optimum MV factorization. However, in general, the two factorizations may lead to different but related results as discussed in Section 4. Notice that, even for non-algebraic factorizations, we can still transform back to the MV domain since all set operations are legal. More discussion on this can be found in Section 5.

As one might expect from the previous example, the procedure of the co-singleton transform is as follows. Given an MV SOP expression, for each literal $x^S$ we replace it with $\prod_{i \notin S} x_i$. To perform the inverse transform, we replace $\prod_{i \in T} x_i$ with $x^{\{j|j \notin T\}}$.

This answers the questions raised at the beginning of this section. However, one might be still curious whether the co-singleton transform is the most compact way.

PROPOSITION 2. *To represent $2^n$ possible literals of an $n$-valued variable, any feasible binary coding has length at least $n$, i.e. requires at least $n$ binary variables.*

Since the co-singleton transform uses exactly $n$ "bits" in this case, it is optimally compact. However, for a particular expression, it is possible that one can rewrite the literals of an $n$-valued variable with fewer bits than $n$.

On the other hand, one might ask whether purely algebraic and semi-algebraic operations are closed under the co-singleton transform. This question can be analyzed as follows. Let $\mathcal{F}$ be a factored form of an MV SOP expression $\mathcal{E}$, and $\mathcal{F}'$ and $\mathcal{E}'$ be the co-singleton transformed versions of $\mathcal{F}$ and $\mathcal{E}$ respectively. Then

THEOREM 1. *If $\mathcal{F}$ is a purely algebraic factorization of $\mathcal{E}$, then $\mathcal{F}'$ is a purely algebraic factorization of $\mathcal{E}'$.*

PROOF. Since $\mathcal{F}$ is a purely algebraic factorization of $\mathcal{E}$, variables in any two product clauses of $\mathcal{F}$ are disjoint by definition. Also as the co-singleton transform is a bijective mapping, "variables" in two different clauses of $\mathcal{F}'$ must be disjoint as well. Hence $\mathcal{F}'$ is purely algebraic. □

However, the converse is not true. Even if $\mathcal{F}'$ is purely algebraic, $\mathcal{F}$ is not necessarily purely algebraic. In fact, it can be derived from $\mathcal{E}$ using only semi-algebraic operations. This can be seen from the previous example. Thus purely algebraic operations are not closed under the co-singleton transform. On the other hand,

THEOREM 2. *If $\mathcal{F}$ is a semi-algebraic factorization of $\mathcal{E}$, then $\mathcal{F}'$ can be derived from $\mathcal{E}'$ using only semi-algebraic operations.*

PROOF. Since $\mathcal{F}$ is a semi-algebraic factorization of $\mathcal{E}$, there exists some variable $x$ that appears in two product clauses $C_1$ and $C_2$ of $\mathcal{F}$. Assume, without loss of generality, that $x^{S_1}$ and $x^{S_2}$ are the two literals in $C_1$ and $C_2$ respectively. Then in $\mathcal{F}'$, the corresponding two clauses $C_1'$ and $C_2'$ are non-disjoint in $x_i$ if and only if $i \notin S_1 \cup S_2$. If there exists such $x_i$, then $\mathcal{F}'$ is a semi-algebraic factorization of $\mathcal{E}'$. Otherwise, $\mathcal{F}'$ is a purely algebraic factorization of $\mathcal{E}'$. □

THEOREM 3. *If $\mathcal{F}'$ is a semi-algebraic factorization of $\mathcal{E}'$, then $\mathcal{F}$ is a semi-algebraic factorization of $\mathcal{E}$.*

PROOF. Because $\mathcal{F}'$ is a semi-algebraic derivation from $\mathcal{E}'$, there exists some "binary variable" $x_i$ that appears in two product clauses. This implies that in $\mathcal{F}$, variable $x$ appears in the corresponding clauses. Therefore $\mathcal{F}$ must be a semi-algebraic derivation from $\mathcal{E}$ □

From Theorems 2 and 3, we know that semi-algebraic operations, which stem from the factorization procedure, are closed under the co-singleton transform. It follows that binary semi-algebraic operations can always be used to simulate MV semi-algebraic operations through the co-singleton transform.

Binary encoding is the process of associating each value of an MV variable with some *distinct*[2] minterm(s) in the space spanned by some binary variables. The $x_i$ that have been introduced can be interpreted this way. For example, the value $i$ can be expressed as

$$x_0x_1 \cdots x_{i-1}\overline{x}_ix_{i+1} \cdots x_{n-1}.$$

Thus the $x_j$ variables are simply the complement of the "1-hot" variables that have been used in the past. However, placing $\overline{x}_i$ explicitly in the expression makes the complemented 1-hot coding unsuitable for the EBD operations. In fact, the co-singleton transform is more than just binary encoding because it implicitly incorporates don't care minimization and thus can map a minterm to different values. A better way to think of the transformed expression is still as a multi-valued expression, but it is restricted to only co-singleton literals. If we think of an MV literal as an OR of some values, then an MV SOP is like a 3-level expression. The co-singleton transform makes this into a two-level expression with the same number of cubes but more literals. Hence a co-singleton transformed expression reflects a direct PLA implementation of an MV SOP, where values of each variable are input as 1-hot signals. These are immediately inverted and appropriate connections are made to form the cubes. Since a PLA is usually implemented as a NOR-NOR, in fact no inverters are really needed. The PLA AND-plane has $n$ columns for an $n$-valued variable. This is similar to what is done with "bit-pairing" some of the binary inputs. For example, combining two binary variables creates a 4-valued signal. This has 4 columns associated with it. After MV minimization, the MV SOP obtained, if represented in "positional" notation, indicates which connections to make in the PLA.

# 4. ALGEBRAIC OPERATIONS

## 4.1 Factorization and Decomposition

One difference between the current MV algebraic operations and the binary ones is the set of divisors used. In the MV algebraic case, only divisors with no "common cube" are considered. An expression has a common cube if there is a literal of some cube that contains all other literals of that variable appearing in the other cubes of the expression. In that case, the dominating literal can be factored out to obtain a factored expression with no increase in literals.

[2]Here we separate the notion of encoding and don't care minimization with respect to unused minterms.

EXAMPLE 1. *The following expression has a common cube.*

$$a^{\{0,1,3\}}b^{\{2\}} + a^{\{1,3\}}b^{\{1\}} = a^{\{0,1,3\}}(b^{\{2\}} + a^{\{1,3\}}b^{\{1\}}).$$

In contrast, the notion of being "cube-free" is that, for each variable, the supercube (literal) of all literals of that variable appearing in that expression is computed. If the supercube is 1 (the universe literal containing all values for that variable) for all variables appearing in the expression, then the expression is cube-free. Making an expression cube-free by factoring out the supercube literals is likely to increase the number of literals in the factored form of the expression.

EXAMPLE 2.

$$a^{\{0,1\}}b^{\{2\}} + a^{\{1,3\}}b^{\{1\}} = a^{\{0,1,3\}}b^{\{1,2\}}(a^{\{0,1,2\}}b^{\{0,2,3\}} + a^{\{1,2,3\}}b^{\{0,1,3\}})$$

*The expression in the parenthesis has been made cube-free.*

Thus in the MV domain the notion of common-cube free replaced the notion of cube-free (used in the binary domain) for algebraic operations[3]. However, making an expression cube-free moves it "nearer" to being prime (which may be good in some sense) since the maximum number of values for each variable is inserted into the expression.

EXAMPLE 3. *The expression* $a^{\{0,1\}}b^{\{2\}}+a^{\{1,3\}}b^{\{1\}}$ *has no common cube in the MV domain and so would be a candidate divisor. However, mapping these two cubes into binary using the co-singleton transform, yields*

$$a_2a_3b_0b_1b_3 + a_0a_2b_0b_2b_3 = a_2b_0b_3(a_3b_1 + a_0b_2)$$
$$= a^{\{0,1,3\}}b^{\{1,2\}}(a^{\{0,1,2\}}b^{\{0,2,3\}} + a^{\{1,2,3\}}b^{\{0,1,3\}})$$

*Thus the corresponding cube-free divisor is*

$$a_3b_1 + a_0b_2 = a^{\{0,1,2\}}b^{\{0,2,3\}} + a^{\{1,2,3\}}b^{\{0,1,3\}}$$

EXAMPLE 4. *Consider a slightly different example,*

$$a^{\{0,1\}}b^{\{2\}} + a^{\{0,1,3\}}b^{\{1\}} = a^{\{0,1,3\}}(a^{\{0,1\}}b^{\{2\}} + b^{\{1\}}).$$

*Thus, the associated* common-cube free divisor *is* $a^{\{0,1\}}b^{\{2\}}+b^{\{1\}}$. *However, the co-singleton transformed expression is*

$$a_2a_3b_0b_1b_3 + a_2b_0b_2b_3 = a_2b_0b_3(a_3b_1 + b_2)$$
$$= a^{\{0,1,3\}}b^{\{1,2\}}(a^{\{0,1,2\}}b^{\{0,2,3\}} + b^{\{0,1,3\}})$$

*Thus, the associated MV* cube-free divisor *is*

$$a_3b_1 + b_2 = a^{\{0,1,2\}}b^{\{0,2,3\}} + b^{\{0,1,3\}}.$$

The difference between these two results is that for the cube-free notion associated with the binary case, the divisor has been "lifted" to have the most values possible. Thus since $a^{\{0,1,3\}}$ has been factored out, the value 2 for $a$ can be inserted everywhere in the cube-free factor. Similarly, the values 0 and 3 can be inserted everywhere for $b$. By doing this lifting, we can transform the common-cube free divisor into the cube-free divisor:

$$a^{\{0,1\}}b^{\{2\}} + b^{\{1\}} \longrightarrow a^{\{0,1,2\}}b^{\{0,2,3\}} + b^{\{0,1,3\}}.$$[4]

[3]In the binary domain, *cube-free* and *common-cube free* are equivalent definitions.

[4]To understand that this is cube free, note that literals with all values are suppressed by convention, e.g. $a^{\{0,1,2,3\}} = 1$ is suppressed in this expression.

In the MV algebraic case, the divisor values are "lowered" as much as possible. Thus the two methods are incomparable. In general it is not clear which divisor is preferable. In fact, there are other divisors between these two extremes (obtained by inserting **some** of the values allowed); one of these may be preferred, since it may coincide with a divisor of another expression. Thus the most general method would be to find all divisors when looking for common divisors. Unfortunately, doing this appears to be too expensive.

EXAMPLE 5.

$$a^{\{2\}}b^{\{1\}} + a^{\{1\}}b^{\{3\}} + a^{\{0\}}b^{\{1,2\}} + a^{\{0,1\}}b^{\{2\}}$$
$$= (a^{\{0,2\}}b^{\{1,2\}} + a^{\{0,1\}}b^{\{2,3\}})(a^{\{1,2\}}b^{\{1,3\}} + a^{\{0,1\}}b^{\{1,2\}})$$

*which translated to the binary domain (and factored further) is*

$$= (a_1b_3 + a_2b_1)(a_0b_2 + a_2b_3)a_3b_0$$

*Note that this is a semi-algebraic factorization. On the other hand, EBD factorization yields:*

$$a_0a_1a_3b_0b_2b_3 + a_0a_2a_3b_0b_1b_2 + a_1a_2a_3b_0b_3 + a_2a_3b_0b_1b_3$$
$$= a_3b_0(a_2b_3(a_1 + b_1) + a_0b_2(a_1b_3 + a_2b_1))$$
$$= a^{\{0,1,2\}}b^{\{1,2,3\}}(a^{\{0,1,3\}}b^{\{0,1,2\}}(a^{\{0,2,3\}} + b^{\{0,2,3\}}) +$$
$$a^{\{1,2,3\}}b^{\{0,1,3\}}(a^{\{0,2,3\}}b^{\{0,1,2\}} + a^{\{0,1,3\}}b^{\{0,2,3\}}))$$

Thus the EBD factorization is different for two reasons. First, no semi-algebraic factorization is done in the EBD method. Second, the EBD result has been lifted to include as many values as possible. The values in the EBD result can be lowered to make it more comparable, by multiplying out the cubes on the outside of the parenthesized expressions (which has the effect of removing values).

It is possible using simple rules to modify one result to be closer to the other. For example, the following rule can be used to convert a binary factorization into one that is more similar to the MV factorization result.

1. In the factorization tree, at a node that is factored as a product of expressions, if a binary literal, $x_i$, has been factored out and one of the expressions has associated literals $x_j$ everywhere, then multiply $x_i$ into all expressions everywhere but leave terms in any expression with no $x_j$ untouched. Remove $x_i$ as a factor.
2. Otherwise leave $x_i$ as a factor, but in any expression where an $x_j$ occurs, replace it with $x_jx_i$.

EXAMPLE 6. *Using the above procedure, the following expression is transformed,*

$$(a_1b_1 + b_2)(a_0b_0 + a_2)a_3b_3 \longrightarrow$$
$$(a_1a_3b_1b_3 + b_2b_3)(a_0a_3b_0b_3 + a_2a_3)$$
$$= (a^{\{0,2\}}b^{\{0,2\}} + b^{\{0,1\}})(a^{\{1,2\}}b^{\{1,2\}} + a^{\{0,1\}}).$$

*Note that* $a_3$ *is not put with* $b_2$ *in the first parenthesis since no other* $a_i$ *occurs there with* $b_2$. *Also, note that when* $a_3$ *and* $b_3$ *are put back, this results in a semi-algebraic factorization (we use* $a_3a_3 = a_3$*). This is the same result obtained by the MV factorization:*

$$= (a^{\{0,2\}}b^{\{0,2\}} + b^{\{0,1\}})(a^{\{1,2\}}b^{\{1,2\}} + a^{\{0,1\}}).$$

EXAMPLE 7.

$$(a_1b_1 + b_2)a_3b_3 \longrightarrow (a_1a_3b_1b_3 + b_2b_3)a_3.$$

*using Rule 2 for* $a_3$.

## 4.2 Algebraic Division

In algebraic division, a divisor, $d$, and an expression, $f$ are given; the problem is to find a quotient, $q$, and remainder, $r$, such that $f = dq + r$ and where $r$ has as few cubes as possible. In the MV case, this is called exact algebraic division[5]. Two algorithms were also defined, *matching* and *satisfiability matrix*; the first was defined for the case where the divisor had only two cubes; it was much faster.

Results obtained by EBD algebraic division and the MV exact division method are comparable but not necessarily identical for the same reasons as for factorization. In both cases the divisor is given. So in the expression $f = dq + r$, $f$ and $d$ are identical in both cases. Thus,

$$f = dq_1 + r_1 = dq_2 + r_2$$

where subscripts 1 and 2 refer to the EBD result and the MV result respectively. For the reasons discussed for factorization, the two quotients, $q_1$ and $q_2$, and the two remainders $r_1$ and $r_2$, may differ. $r_2$ may be smaller than $r_1$ because the semi-algebraic division may be able to absorb more cubes in the product; the quotients can differ also because $q_1$ is maximally raised and $q_2$ is maximally lowered.

## 4.3 Common Divisor Extraction

EXAMPLE 8. *This is an example of two functions, where a common factor is found in the transformed binary domain and not in the MV domain. The factorization in the MV domain,*

$$\begin{aligned} f &= a^{\{3\}}b^{\{2,3\}} + a^{\{2,3\}}b^{\{1,3\}} + a^{\{1\}}b^{\{2\}} + a^{\{1,2\}}b^{\{1\}} \\ &= (a^{\{1,3\}}b^{\{2,3\}} + b^{\{1,3\}})(a^{\{2,3\}} + a^{\{1,2\}}b^{\{1,2\}}) \\ g &= a^{\{0,3\}}b^{\{1,3\}} + a^{\{0,2,3\}}b^{\{2,3\}} + a^{\{0,1\}}b^{\{1\}} + a^{\{0,1,2\}}b^{\{2\}} \\ &= (a^{\{0,1,3\}}b^{\{1,3\}} + b^{\{2,3\}})(a^{\{0,2,3\}} + a^{\{0,1,2\}}b^{\{1,2\}}) \end{aligned}$$

*However transformed into the binary domain we get*

$$\begin{aligned} f &= a_0a_1a_2b_0b_1 + a_0a_1b_0b_2 + a_0a_2a_3b_0b_1b_3 + a_0a_3b_0b_2b_3 \\ &= (a_3b_3 + a_1)(a_2b_1 + b_2)a_0b_0 \\ g &= a_1a_2b_0b_2 + a_1b_0b_1 + a_2a_3b_0b_2b_3 + a_3b_0b_1b_3 \\ &= (a_3b_3 + a_1)(a_2b_2 + b_1)b_0 \end{aligned}$$

*which has the common factor*

$$a_3b_3 + a_1 = a^{\{0,1,2\}}b^{\{0,1,2\}} + a^{\{0,2,3\}}.$$

EXAMPLE 9. *The expressions,*

$$\begin{aligned} f &= a^{\{2\}}b^{\{1\}} + a^{\{1\}}b^{\{3\}} + a^{\{0\}}b^{\{1,2\}} + a^{\{0,1\}}b^{\{2\}} \\ &= (a^{\{0,2\}}b^{\{1,2\}} + a^{\{0,1\}}b^{\{2,3\}})(a^{\{1,2\}}b^{\{1,3\}} + a^{\{0,1\}}b^{\{1,2\}}) \\ g &= (a^{\{0,2\}}b^{\{1,2\}} + a^{\{0,1\}}b^{\{2,3\}})(c^{\{1\}} + d^{\{1\}}) \end{aligned}$$

*have a common factor in the MV domain but none using the EBD method.*

There are examples where both expressions have no common algebraic factors when factored in either the MV domain or the transformed binary domain, but by selectively adding and deleting values a common factor can be obtained.

Another difference happens when a divisor is extracted and algebraically divided into other expressions. The results can be different between the EBD extraction and the MV extraction e.g. the extraction may result in new nodes $y = k$ and $\tilde{f} = yq + r$ where the EBD $q$ and the MV $q$ may differ.

[5]Inexact division was also defined for the MV case, where given $d$ we seek a better result of the form $f = \tilde{d}\tilde{q} + \tilde{r}$ where $d \subseteq \tilde{d}$.

# 5. NON-ALGEBRAIC OPERATIONS

The interpretation of $\overline{a}_i$ is $a^{\{i\}}$. If we take a co-singleton-transformed binary expression from an MV expression, the new "binary variables" (co-singleton literals) occur in only positive form, i.e. the binary expression is positive unate in these variables.

EXAMPLE 10.

$$a_1b_1 + a_2b_2 = a^{\{0,2,3\}}b^{\{0,2,3\}} + a^{\{0,1,3\}}b^{\{0,1,3\}}$$

*Using $\overline{x}_i\overline{x}_j = \emptyset$, the complement of the left-hand side is*

$$(\overline{a}_1 + \overline{b}_1)(\overline{a}_2 + \overline{b}_2) = \overline{a}_1\overline{b}_2 + \overline{b}_1\overline{a}_2$$

*which translates into $a^{\{1\}}b^{\{2\}} + a^{\{2\}}b^{\{1\}}$. The complement of the right-hand side is*

$$(a^{\{1\}} + b^{\{1\}})(a^{\{2\}} + b^{\{2\}}) = a^{\{1\}}b^{\{2\}} + a^{\{2\}}b^{\{1\}}.$$

Thus we get equal results in this example. However in general this is not always the case.

EBD manipulations, like simplification, yield equivalent results to the MV versions, but EBD results may not be prime and irredundant.

EXAMPLE 11.

$$x^{\{1,2\}}y^{\{0\}} + x^{\{1\}} + x^{\{0,1\}}y^{\{1\}} = x_0y_1 + x_0x_2 + x_2y_0$$

*is irredundant in the binary domain, but is redundant when translated to the MV domain.*

Also, being prime does not translate between domains, because in the binary domain, the relations $\overline{x}_i\overline{x}_j = \emptyset$ are unknown, since $x_i$ and $x_j$ are treated as independent variables in the binary domain. Thus in expanding a cube until it just meets the offset in order to generate a prime, we might not expand far enough in the binary domain because there are additional (unknown) cases where intersections are empty.

If the relation, $\overline{x}_i = x_0x_1 \cdots x_{i-1}x_{i+1} \cdots x_{n-1}$ is used, then every expression is positive unate in the new binary variables. But again, the relation $x_0x_1 \ldots x_{n-1} = \emptyset$ is unknown. Of course, this information could be provided by giving

$$\sum_{i,j}(\overline{x}_i\overline{x}_j),\ i \neq j$$

as don't cares, but then the minimization process becomes more cumbersome[6].

In general, single cube containment is equivalent between both domains, i.e. $C^i \subseteq C^j$ if and only if $B(C^i) \subseteq B(C^j)$, where $B(\cdot)$ converts an MV cube to a binary one via the co-singleton transform.

Although not minimal, EBD Boolean operations may be useful since the EBD result could be improved by a single step of making cubes prime. The result is already minimal with respect to single cube containment. Optionally further redundancy can be removed. The EBD Boolean operation may be faster since in the binary domain the `nocomp` option, which uses the concept of the reduced offset [9], is available. Thus EBD Boolean operations can give a type of sub-optimal MV operations which are possibly faster.

[6]This might be useful and practical, but some experimentation needs to be done here.

**Table 1: Comparison of EBD and MV scripts**

| circuit | EBD time | EBD lits-ff | MV time | MV lits-ff |
|---|---|---|---|---|
| vg2 | 2.9 | 87 | 2.6 | 85 |
| sse | 2.1 | 128 | 2.2 | 120 |
| b12 | 2.4 | 70 | 2.3 | 70 |
| cht | 1.8 | 163 | 1.9 | 164 |
| sqrt8 | 1.1 | 67 | 1.2 | 56 |
| clip | 5.3 | 134 | 7.6 | 129 |
| duke2 | 10.7 | 497 | 24.6 | 488 |
| sand | 23.6 | 545 | 47.5 | 525 |
| f51m | 1.8 | 108 | 2.4 | 97 |
| sao2 | 2.4 | 109 | 4 | 110 |
| term1 | 5.2 | 147 | 6.2 | 142 |
| 9sym | 3 | 72 | 4.6 | 120 |
| alu2 | 12.5 | 266 | 19.4 | 278 |
| sct | 1.9 | 83 | 2 | 90 |
| t481 | 14.2 | 36 | 63.9 | 40 |
| ttt2 | 3.1 | 233 | 4.4 | 221 |
| bw | 3.2 | 194 | 4.4 | 194 |
| rd84 | 5.3 | 87 | 9.5 | 106 |
| squar5 | 1.4 | 58 | 1.4 | 58 |
| z4ml | 1.2 | 38 | 1.4 | 38 |
| C432 | 46.3 | 185 | 49.3 | 195 |
| planet | 24.7 | 605 | 63.5 | 611 |
| vda | 32.3 | 763 | 96 | 777 |
| cps | 93.3 | 1479 | 364.1 | 1524 |
| dk16 | 7.0 | 248 | 9.3 | 238 |
| s953 | 18.6 | 510 | 29.6 | 516 |
| k2 | 269.2 | 1426 | 3351 | 1428 |
| balance | 8.1 | 182 | 84.1 | 217 |
| conv35cc | 1.2 | 83 | 1 | 72 |
| employ1 | 1.7 | 42 | 1.5 | 36 |
| mm3 | 0.9 | 23 | 0.8 | 23 |
| mm5 | 5.3 | 137 | 8.5 | 130 |
| pal3x | 4 | 114 | 4.5 | 100 |
| aluack | 1.4 | 91 | 1.4 | 76 |
| iris | 1.3 | 12 | 1.3 | 12 |
| mm4 | 2 | 75 | 2.3 | 60 |
| monks2tr | 1.2 | 51 | 1.2 | 43 |
| monksltr | 0.9 | 7 | 0.8 | 7 |
| sleep | 36.6 | 33 | 63.7 | 37 |
| car | 1.2 | 43 | 1.3 | 44 |

# 6. EXPERIMENTAL RESULTS

These ideas have been implemented in MVSIS and the MV algebraic methods compared with the EBD algebraic methods in the setting of optimizing multi-valued multi-level networks. For the benchmarks tested, an identical sequence of operations[7] was applied, except in the EBD version, called EBD_script, all MV algebraic operations of MV_script were replaced with their EBD counterparts. Each EBD algebraic operation consists of

1. converting the MV expression(s) to their co-singleton counterpart(s),
2. applying the corresponding binary algebraic operation, and
3. converting the resulting co-singleton form(s) back to MV expression(s).

The results are shown in Table 1 where EBD refers to running the EBD_script and MV refers to running the MV_script. Time is in seconds and `lits-ff` refers to the total number of literals in the factored forms of the MV expressions in a circuit. The last set of examples are multi- valued.

As expected, the results show that quality is essentially maintained by the EBD operations, but the speed is greatly improved for the larger examples. In some benchmarks, a slight loss of quality occurs in using the EBD operations. This is possibly because there are no semi-algebraic operations in the binary implementation. Also, `EBD_fx` extracts common cubes, and some of these are of the type $\prod_i x_i$, which in the MV-domain is just a single literal cube and of no value as a separate node in the network.

[7]The basic script used was an improved multi-valued version of `script.rugged` used in SIS.

# 7. CONCLUSIONS

The co-singleton transform places most MV operations (which include EBD ones) on an equal footing with the binary ones in terms of speed and quality of results. Thus it is an important step towards our goal of making MVSIS the system of choice for optimizing multi-level networks, whether the network is MV or binary. Since MVSIS also includes a number of new ideas that are not in SIS, the quality of the results already exceeds those obtained in SIS. We have also observed that the ability of MVSIS to enter and leave the MV domain during an optimization run, allows more freedom in finding better optimizations and encourages new ideas.

We have not experimented with the MV Boolean operations, mentioned in Section 5, since the EBD Boolean operations would lead to sub-optimal results. In addition, the MV Boolean operations are almost as fast as their binary counterparts. The one exception is that there is no code for the MV counterpart of the reduced offset in ESPRESSO-MV even though the theory exists [9]. The reduced offset helps in those cases where the complement of a large function is needed in order to expand to primes during a two-level SOP optimization. During node minimization, it is common to derive a large don't care set, and a subsequent call to ESPRESSO-MV requires complementation of the onset plus don't care set. However, we intend to experiment with using EBD node-minimization as a technique for trading lower quality for increased speed.

As shown in Section 3, in principle through the co-singleton transform, **all** MV semi-algebraic operations could be reproduced in the binary domain if the binary operations had semi-algebraic extensions. This result suggests that semi-algebraic operations in the binary domain should be revisited in the future.

# 8. REFERENCES


[1] R. K. Brayton. Algebraic methods for multi-valued logic. Technical Report UCB/ERL M99/62, University of California, Berkeley, Dec. 1999.

[2] R. K. Brayton and et al. MVSIS. `http://www-cad.eecs.berkeley.edu/mvsis/`.

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

[4] S. Devadas and A. R. Newton. Exact algorithms for output encoding, state assignment, and four-level boolean minimization. *IEEE Trans. on Computer-Aided Design*, 10(1):13–27, Jan. 1991.

[5] M. Gao and R. K. Brayton. Semi-algebraic methods for multi-valued logic. In *Proc. of the Int'l Workshop on Logic Synthesis*, pages 73–80, May 2000.

[6] M. Gao and R. K. Brayton. Multi-valued multi-level network decomposition. In *Proc. of the Int'l Workshop on Logic Synthesis*, pages 254–260, June 2001.

[7] T. Kam, T. Villa, R. K. Brayton, and A. Sangiovanni-Vincentelli. Multi-valued decision diagrams: Theory and applications. *Int'l Journal on Multiple-Valued Logic*, 4(1-2):9–62, 1998.

[8] L. Lavagno, S. Malik, R. Brayton, and A. Sangiovanni-Vincentelli. MIS-MV: Optimization of multi-level logic with multiple-valued inputs. In *Proc. of the Int'l Conf. on Computer-Aided Design*, pages 560–563, Nov. 1990.

[9] A. Malik, R. K. Brayton, and A. Sangiovanni-Vincentelli. A modified approach to two-level logic minimization. In *Proc. of the Int'l Conf. on Computer-Aided Design*, pages 106–109, Nov. 1988.

[10] H.-M. Wang, C.-L. Lee, and J.-E. Chen. Algebraic division for multi-level logic synthesis of multi-valued circuits. In *Proc. of the Int'l Symp. on Multiple-Valued Logic*, pages 44–51, May 1994.