Semi-Algebraic Methods for Multi-Valued Logic

Minxi Gao and Robert K. Brayton
Electrical Engineering and Computer Sciences Dept.
University of California, Berkeley CA 94720
({minxi,brayton}@eecs.berkeley.edu)

May 1, 2000

Abstract

We give several semi-algebraic methods for manipulating multi-valued logic functions. The methods treat binary and multi-valued variables uniformly. They include methods for finding common sub-expressions, semi-algebraic division, decomposing a multi-valued network, and factoring an expression. Even in some binary cases, the methods generalize algebraic mathods for binary logic synthesis [2]. The algorithms have been implemented in a system called MV-SIS for optimizing multi-valued logic and tested for quality on a small set of examples. Their speed and quality seem sufficient for filling the role played by algebraic division in binary logic synthesis.

1 Introduction

Multi-valued (MV) logic synthesis can be useful in the following applications:

1. initial manipulation of a hardware description before it is encoded into binary and processed by standard logic synthesis programs; MV is a natural way to describe procedures at a higher level,
2. a front end to a software compiler, since software lends itself naturally to the evaluation of multi-valued variables in a single cycle,
3. in some asynchronous applications.

Although there has been a lot of work on multi-valued logic manipulation and optimization, one set of techniques missing so far is the algebraic methods which are at the core of efficient binary logic synthesis.

Algebraic methods are used after first casting the logic expression into a minimized sum-of-products. Then the result is manipulated as an algebraic expression, ignoring the Boolean identities $x\overline{x} = 0, xx = x, x + \overline{x} = 1$. The intuition is that if two functions have a common subexpression or divisor, then often this can be recognized from their minimized sum-of-products expressions. This results in faster methods for manipulating the logic, such as factoring and finding common divisors. Although some optimality is lost, this can be recovered by using Boolean methods later.

In this paper, we develop, more fully, algebraic type methods for MV-logic. The basis for these ideas originate in the paper of Lavagno et. al. [3].

2 Notation

In general, a MV-logic function can have MV input variables and an MV output. A function with a single binary output is called an *MV-function*.

A function with k output values can be represented by k MV-functions or as a single function with k output values or as multiple binary output functions where the output values are encoded. All such representations have multi-valued inputs.

An **MV-network** is a network of nodes; each node represents a function with a single multi-valued output. There is one MV variable associated with the output of each node. An edge connects node i to node j if the function at j depends explicitly on the variable associated with node i, typically denoted, y_i. The network has a set of primary inputs and a set of nodes which are designated as the outputs of the network. An intermediate format for representing such a network is

BLIF-MV used in the VIS system [1].

In general, a variable x_i is multi-valued and takes on values from the set $P_i = \{0, 1, ..., |P_i|-1\}$. A **literal** of an MV-variable x is associated with a subset of values for that variable. For example, suppose x can take on 5 values $\{0, 1, 2, 3, 4\}$. Then $x^{\{0,2\}}$ and $x^{\{1,2,4\}}$ are literals of x. The interpretation of $x^{\{0,2\}}$ is that it is a binary logic function which is 1 if x has either the value of 0 or 2, and 0 otherwise. Note that $x^{\{0,1,2,3,4\}} = 1$ since all five possible values appear in the literal. A **product term** or **cube** is a conjunction of literals and evaluates to 1 if and only if each of the literals evaluates to 1. Additionally, a cube can be thought of as simply a set of values. We use the notation $\tilde{c}$ to denote the cube consisting of all values **not** in the cube c. A **sum-of-products** (SOP) is the disjunction of a set of product terms. It evaluates to 1 if any of the products evaluates to 1. Note that a SOP is a function with a single binary output and multiple multi-valued input variables.

The **supercube** of a set of cubes f (a SOP or an "logic expression"), denoted $\sigma(f)$, is the smallest cube containing f. It is the cube formed by taking the union of all the values in all the cubes. Similarly, $\tilde{\sigma}(f)$ is the cube consisting of the set of values not in $\sigma(f)$. Note that in general, $\tilde{\sigma}(f) \neq \overline{\sigma(f)} \neq \sigma(\overline{f})$.

The **cofactor** of a set of cubes d with respect to a cube c, denoted d_c, is the set of cubes obtained by eliminating cubes of d that do not intersect c, and then adding to each remaining cube, those values not in the cube c, i.e. the values in $\tilde{c}$.

Example 1 *If x and y each have 5 values, $d = x^{\{0,1,4\}} + y^{\{1,2,4\}} + x^{\{0\}}y^{\{0,3\}}$ and $c = x^{\{1,2\}}y^{\{1,3\}}$, then $d_c = x^{\{0,1,3,4\}} + y^{\{0,1,2,4\}}$.*

Definition 1 *An expression f is* **cube-free** *if $\sigma(f) = 1$.*

Example 2 *$f = a^{\{1,3\}}b^{\{2,3\}} + a^{\{0,3\}}b^{\{1,3\}}$ is not cube-free, $\sigma(f) = a^{\{0,1,3\}}b^{\{1,2,3\}} \neq 1$.*

Definition 2 *An expression has a* **common cube** *if for each variable, there is* **no** *literal (except the literal 1) appearing in the cubes of the expression that contains all other literals of that variable in the expression.*

Example 3 *$g = a^{\{1,3\}}b^{\{1,2,3\}} + a^{\{0,1,3\}}b^{\{1,3\}}$ has a common cube, $a^{\{0,1,3\}}b^{\{1,2,3\}}$, because the literal $a^{\{0,1,3\}}$ contains $a^{\{1,3\}}$ and $b^{\{1,2,3\}}$ contains $b^{\{1,3\}}$.*

Note that the expression, $f = a^{\{1,3\}}b^{\{2,3\}} + a^{\{0,3\}}b^{\{1,3\}}$, has no common cube even though $f \subseteq a^{\{0,1,3\}}b^{\{1,2,3\}}$. We want to make a distinction here because if we factor out $a^{\{0,1,3\}}b^{\{1,2,3\}}$ from f to obtain $f = a^{\{0,1,3\}}b^{\{1,2,3\}}(a^{\{1,2,3\}}b^{\{0,2,3\}} + a^{\{0,2,3\}}b^{\{0,1,3\}})$ we obtain a cube-free expression in the parenthesis, but it is not simpler.

2.1 Factoring Out a Common cube

If there is a common cube of an expression, we can factor it out and obtain a simpler expression in several ways.

1. cofactor the expression by the common cube (e.g. $a^{\{1,3\}}b^{\{1,2,3\}} + a^{\{0,1,3\}}b^{\{1,3\}} = a^{\{0,1,3\}}b^{\{1,2,3\}}(a^{\{1,2,3,4\}} + b^{\{0,1,3,4\}})$),
2. remove all literals appearing in the common cube (e.g. $a^{\{1,3\}}b^{\{1,2,3\}} + a^{\{0,1,3\}}b^{\{1,3\}} = a^{\{0,1,3\}}b^{\{1,2,3\}}(a^{\{1,3\}} + b^{\{1,3\}})$).

In both cases the expression in the parenthesis is simpler in some sense. In the first case, values (not in the common cube) are added to the expression. In the second case, extra values are added but only to make the common cube literals equal 1 (for example in the second case, we added values to $a^{\{0,1,3\}}$, namely $\{0, 4\}$ to make the resulting literal 1). In this paper, we use the second method of making an expression simpler by factoring, in order to keep the number of values in an expression minimal. In fact, the two cases represent upper and lower bounds for inserting values in the factored results. The values not in the common cube are called **redundant values** and we can use them like don't cares to obtain many cube-free expressions. Thus, unlike the binary case, the cube-free expression is not unique. This makes the problem of finding common divisors among two or more expressions more difficult than in the binary case.

Example 4 *Consider the following two expressions.*

$$\begin{aligned} f &= x^{\{0,1,3\}}(y^{\{1,2\}} + x^{\{0\}}y^{\{0\}}) \\ g &= x^{\{0,2,4\}}(y^{\{1,2\}} + x^{\{0,2,3\}}y^{\{0\}}) \end{aligned}$$

At first glance, f and g seem to have no common divisor. However, inside the parentheses of f, 2 and 4 are redundant values for x, and inside the parentheses of g, 1 and 3 are redundant values for x. If we choose to include 2 in the first, and remove 3 in the second, we get

$$\begin{aligned} f &= x^{\{0,1,3\}}(y^{\{1,2\}} + x^{\{0,2\}}y^{\{0\}}) \\ g &= x^{\{0,2,4\}}(y^{\{1,2\}} + x^{\{0,2\}}y^{\{0\}}), \end{aligned}$$

yielding the common divisor of $y^{\{1,2\}} + x^{\{0,2\}}y^{\{0\}}$.

3 Satisfiable Matrices

The work in [3] related the factorization of a function of a single multi-valued variable with the existance of a "satisfiable" matrix of MV-literals. We generalize the definitions and procedures of [3]:

- In general all variables can be multi-valued.
- The elements in the matrix are MV-cubes.
- The rows and columns are not associated with binary co-kernel and kernel cubes.

Definition 3 *Consider any rectangular arrangement of a set of MV-cubes. It is* **satisfiable** *if for all values of all variables, each value satisfies the following condition: Let I be the set of rows and J the set of columns in M in which value v appears. Then value v satisfies the* **value condition** *if it appears in all entries of M given by $\{M_{i,j} | i \in I, j \in J\}$.*

Example 5 *The matrix of values*

1,2	*1,3*	*2,3*
2,3	*1,4*	*4*

is not satisfiable, since it should have a 3 in all entries, a 2 in (2,3) and a 1 in (2,1).

Note that this definition applies equally to binary as well as MV-variables.

Similar to the procedure in [3], one can derive a product of two expressions from a satisfiable matrix:

1. For each row i, form $e_{r,i}$, the supercube of all cubes in that row.
2. OR these together to form the *row expression*, $e_r = \sum_i e_{r,i}$.
3. For each column j, form $e_{c,j}$, the supercube of all cubes in that column.
4. OR these together to form the *column expression*, $e_c = \sum_j e_{c,j}$.

Theorem 1 *If M is a satisfiable matrix, then $\sum_{i,j} M_{ij} = (e_r)(e_c)$, i.e. the SOP M can be re-written as a product of two expressions.*

Proof. In fact, we claim that

$$M_{ij} = e_{r,i} \cap e_{c,j}.$$

Clearly $M_{ij} \subseteq e_{r,i} \cap e_{c,j}$ because by definition, $e_{r,i}$ is a cube containing all cubes in row i and $e_{c,j}$ is a cube containing every cube in column j. Now suppose that $M_{ij} \not\supseteq e_{r,i} \cap e_{c,j}$. Then there exists a variable with a value v such that $v \in e_{r,i} \cap e_{c,j}$ but $v \notin M_{ij}$. However, v must be in M_{ik} for some k (since $v \in e_{r,i}$), and also $v \in M_{mj}$ for some m (since $v \in e_{c,j}$). Therefore, by the value condition for a satisfiable matrix, $v \in M_{ij}$ (as well as $v \in M_{mk}$), a contradiction. Hence, $M_{ij} \supseteq e_{r,i} \cap e_{c,j}$. □

3.1 Finding a Large Satisfiable Matrix

We give a method, given a set of cubes, for finding a subset that can be rearranged into a (largest) satisfiable matrix. The method is based on pre-selecting the number of rows in the matrix. Then the array is found by a branch and bound technique. Entries are selected in the matrix in column order, i.e. cubes are selected one at a time for $M_{11}, M_{21}, ..., M_{12}, ...$ in that order. At each point, the entries selected are guaranteed, thus far, to satisfy the value condition against the previously selected cubes.

3.1.1 Lower and Upper Bounds

We require that the selected cube c for the next entry, M_{ij}, should satisfy a lower bound cube l^{ij} and two upper bound cubes, u_1^{ij} and u_2^{ij}, i.e.

$$l^{ij} \subseteq c \subseteq u_1^{ij} \cap u_2^{ij}$$

We now define these bounds and prove that they characterize a satisfiable matrix precisely. These bounds are used in an efficient branch and bound algorithm.

Let (i, j) be the matrix position for the next entry to be selected.

Lower Bound Cube: The cube l^{ij} (which depends on i, j) consists of the following set of values:

$$\begin{aligned} l^{ij} &= \{v \mid (\exists (k < j), v \in M_{i,k}) \\ &\quad \text{and } (\exists (m < i), v \in M_{m,j})\} \end{aligned}$$

Note that for $i = 1$ or $j = 1$, this is the null set.

Upper Bound Cube 1: The cube u_1^{ij} (which depends on i, j) consists of the following set of values:

$$\begin{aligned} u_1^{ij} &= \{v \mid (\exists (k < j), v \in M_{i,k}) \\ &\quad \text{or } (\forall (n \neq i, m < j), v \notin M_{n,m})\} \end{aligned}$$

Note that for $j = 1$, this is set of all values.

Upper Bound Cube 2: The cube u_2^{ij} (which depends on i, j) consists of the following set of values:

$$\begin{aligned} u_2^{ij} &= \{v \mid (\exists (n < i), v \in M_{n,j}) \\ &\quad \text{or } (\forall (m < j) \forall (n < i), v \notin M_{n,m})\} \end{aligned}$$

Note that for $j = 1$, this is the set of all values.

Theorem 2 *A matrix of cubes M is satisfiable if and only if for each (i, j), $M_{i,j}$ satisfies $l^{ij} \subseteq M_{i,j} \subseteq u_1^{ij} \cap u_2^{ij}$.*

Proof. Omitted for brevity. □

We have also developed some additional techniques that help prune the search and make the construction of satisfiable matrices much faster.

4 Semi-Algebraic Division

One of the applications of the above search process is to factor an expression, i.e. given an expression (sum-of-products), d, which will serve as the divisor, and another expression f to be factored, find the dividend, e, i.e. a largest expression e, such that $f = de + r$. In this equation, each side is a set of cubes and equality means that the two sets are equal. de produces a set of cubes of size $|d| \times |e|$, i.e. the cross product of the two sets. The product of two cubes is the cube containing the intersection of the two sets of values for all the variables.

We relax the definition of algebraic product for MV-variables in that we *do not require that the two expressions d and e have disjoint sets of variables.*

Example 6 *Consider the product*

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

When this is multiplied out, we get

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

Although it did not happen in this example, null cubes could be produced. Note that we use non-algebraic properties in performing this product since the set of values obtained for a variable is the intersection of the two sets from each cube, e.g.

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

which is analogous to using $xx = x$ for the binary case.

In our semi-algebraic division algorithms, we start with a given divisor d and search for a satisfiable matrix M formed from a subset of the cubes of f. The row expression e_r associated with M will be related to d. We look at two types of semi-algebraic division methods, exact and inexact. In **exact division**, we require that $du = e_r$ where u is some cube. In **inexact division** the requirement is relaxed to $e_{r,i} \subseteq d_i$ and thus $e_r \subseteq d$.

In exact division, the column expression, e_c, is the dividend, and the cubes of f not included in $M = (e_r)(e_c) = due_c$ form the remainder r. Each cube of d is associated with a row.

In both types of division, each cube placed in a row must be contained in the associated cube of d. Thus we can restrict the search for cubes in that row to cubes $M_{ij} \subseteq d_i$. In this way, the cubes of d serve to limit the search and hence make it more efficient.

Example 7 *(inexact division) Let* $d = a + b$ *and*

$$f = a\overline{b}x + a\overline{b}y + \overline{a}bx + \overline{a}by$$

We get $e_r = a\overline{b} + \overline{a}b$ *and* $e_c = x + y$.

Note that the divisor, d, just seeds the search; d does not have to be necessarily a algebraic divisor of f.

Example 8 *(exact division) Let* $d = a\overline{b} + \overline{a}b$ *and*

$$f = a\overline{b}xz + a\overline{b}yz + \overline{a}bxz + \overline{a}byz.$$

We get $e_r = z(a\overline{b} + \overline{a}b)$.

In general, we can use any expression to start the inexact process, even, for example, $d = 1 + 1 + 1$, in which case d has no information and we are just looking for a largest satisfiable matrix with 3 rows.

There are several applications where we want the row expression $e_r = du$. For example, in decomposition, a common divisor d is extracted from a set of functions and implemented as a separate function. Then d is used to rewrite certain functions, f_i, in terms of a new variable $y = d$: $f_i = ye_i + r_i = de_i + r_i$. It would not be acceptable to have a result where each row expression is not d ANDed with some cube u_i, i.e. $f_i = q_ie_i + r_i$, where each $q_i \subseteq d$, but $q_i \neq du_i$. If $q_i \neq du_i$ then d would not appear (algebraically[1]) in each of the f_i.

However, in an application like factoring, we use the divisor in only one place.

Example 9 *Consider the function,*

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

Suppose we determine from looking at cubes 1 and 3 above that we want to divide by

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

If we require that the row expression equal this, then we get $c^{\{3\}}$ *as the dividend. However, if we only use* $a^{\{0,1,2\}} + b^{\{1,2,3\}}$ *as a seed and do inexact division, we can get a larger satisfiable matrix and achieve the factorization*

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

Again the row expression is contained in the original divisor, but is not equal to du.

Because exact division is more constrained than inexact division, one can expect to find tighter bounds on the entries in the matrix. We have derived such bounds and have given a conjecture on the tightness of these bounds. Although these bounds would make exact division more efficient, the following direct method seems even more promising.

4.1 A Direct Method for Exact Division

We briefly describe a direct (non-search) method (see Appendix for more details). A set of candidate dividend cubes S^{d_i} is associated with each cube d_i of the divisor d. Unlike the binary case, a dividend cube is not unique; given a cube d_i of the divisor and a cube $c_j \subseteq d_i$ of the function f, an associated set of dividend cubes is given by[2]

$$c_j \subseteq k_{ij} \subseteq \sigma(c_j + \tilde{d}_i).$$

Like the binary case, the sets are "intersected" to find the subset of dividend cubes common to all $\{S^{d_i}\}$. A cube k is in this intersection if and only if there exists a set $J = \{j_i, i = 1, \ldots, |d|\}$ such that

$$c_{j_i} \subseteq k \subseteq \sigma(c_{j_i} + \tilde{d}_i), \forall j_i \in J.$$

Since the above inequalities can be represented as a cube K_{ij} in a larger space, common intersection is equivalent to pairwise intersection, i.e. a common cube exists among $\{K_{i,j_i} | i = 1, \ldots, |d|\}$ if and only if each pair $K_{i_1,j_{i_1}}$ $K_{i_2,j_{i_2}}$ intersects. The algorithm allows a cube c_j to participate in several of the S^{d_i} i.e. we allow for duplication of cubes in f. The set of intersecting cubes $\{k\}$ is the resulting dividend.

There is an interesting connection with inexact division, namely let e_i be the final cube in row i after

[1] Since $q_i \subseteq d$, there exists a function g_i such that $q_i = g_id$, hence $f_i = dg_ie_i + r_i$, so d is a Boolean divisor of f_i. However, the combination g_ie_i may be more complex than the original function f_i.

[2] In the binary case of algebraic division, the candidate dividend cubes for d_i are those cubes k_{ij} where $c_j = d_ik_{ij}$ and $c_j \in f$, i.e. $k_{ij} = c_{j d_i}$. Thus for each $c_j \subseteq d_i$, $k_{i,j}$ is unique.

inexact division. Then

$$c_j \subseteq k_{ij} \subseteq \sigma(c_j + \tilde{d}_i) \subseteq \sigma(c_j + \tilde{e}_i)$$

Neither the direct method nor the improved search for exact division using the tighter bounds have been implemented yet. Since we believe that the direct method will be the most efficient, this is being implemented currently.

5 Factorization

The divisor d is used only to focus and limit the search process for a satisfiable matrix. The row expression obtained from the satisfiable matrix need not equal du. In this case we obtain $f = \hat{d}e_c + r$ which can be an acceptable start for factoring f. Additionally, just as in the binary case, it is not necessary to get the best result at first; as a second step, e_c with its common cube extracted, can be used as the divisor in a second division, leading possibly to a better factorization. This is the basis of *quick factor* (QF), used in SIS [4], where the first divisor is chosen to be a level-zero kernel.

We implemented such a factoring process, MV-QF:

1. The first seed divisor is chosen to be the first pair of cubes of f found that have a common cube, in analogy to the method of [5].

2. The literals of this common cube are extracted from the cube pair (by making it cube-free, according to Method 2 in Section 2.1).

3. This cube-free expression is used as the first candidate divisor.

4. After this division is preformed, the common cube of the dividend is extracted and used as the divisor in a second division to get an partial factorization.

5. Each divison, quotient and remainder is factored recursively to get the final factorization.

Example 10 *(factorization produced by the MV-QF)*

$$\begin{aligned} S3 \;=\; & (a^{\{0,1\}} + c^{\{1,4\}} + a^{\{2\}}b^{\{1,2,3\}}c^{\{3,4\}}) \\ & ((f^{\{1,2\}} + f^{\{0,2\}}g^{\{1\}})e^{\{1\}}g^{\{1,2\}} + \\ & f^{\{2\}} + d^{\{0,2,4,5\}}g^{\{1,2,3\}} + \\ & d^{\{0,1,2,4,5\}}f^{\{0,1\}}g^{\{3\}} + d^{\{2,3,4,5\}}e^{\{0\}}g^{\{2\}}) \end{aligned}$$

Note that even though the first divisor used in MV-QF consisted of only two cubes, by using the column expression (with its common cube extracted) as the second divisor, we obtain a much stronger factorization.

6 A Kerneling Process

We seek a type of kerneling process similar to that used for binary functions [2, 5]. Kerneling, used in decomposition, is an efficient way to identify common divisors among a set of expressions. We follow a process analogous to the two-cube divisor method of [5]. For the binary case, the **two-cube divisors** of an expression is the set

$$\tau(f) = \{\text{cube\_free}(c_i, c_j) | i \neq j, c_i, c_j \in f\}$$

Note that there are at most $\frac{|f|(|f|-1)}{2}$ two-cube divisors of an expression f, where $|f|$ is the number of cubes in f. This is the method implemented as *fast_extract* in SIS.

In the method we propose, a divisor is given a figure of merit by keeping track of the number of times it or its complement exact divides into all the expressions being considered. In decomposition, the divisor with the greatest merit is chosen, implemented as a separate function, and substituted into all the expressions in which it appears. The substitution is performed using exact semi-algebraic division. Once a network has been decomposed by this process, functions can be selectively eliminated if their final figure of merit in implementing the network is below a given theshhold.

Example 11 *Consider the following decomposed network, where the numbers in the parentheses of the input variables give the number of values for the variable. The y_i subexpressions were not in the original description, and are the result of the kerneling process followed by selective elimination of expressions.*

.in	$a(3), b(4), c(5), d(6), e(2), f(3), g(4)$
.out	$S3, S6$

$$
\begin{aligned}
S3 &= y_1^{\{1\}}(g^{\{12\}}y_4^{\{1\}} + y_2^{\{1\}} + d^{\{01245\}}g^{\{3\}}) \\
S6 &= (y_2^{\{1\}} + d^{\{01245\}}f^{\{01\}}g^{\{3\}})a^{\{01\}} + g^{\{12\}} \\
&\quad ((d^{\{0245\}} + f^{\{2\}})y_0^{\{1\}} + y_1^{\{1\}}y_4^{\{1\}}) \\
y_0 &= b^{\{123\}}c^{\{3\}} + c^{\{14\}} \\
y_1 &= a^{\{01\}} + y_0^{\{1\}} \\
y_2 &= d^{\{0245\}}g^{\{123\}} + f^{\{2\}} \\
y_4 &= (f^{\{1\}} + g^{\{1\}})e^{\{1\}} + d^{\{2345\}}e^{\{0\}}g^{\{2\}}
\end{aligned}
$$

Note that even though only two-cube divisors were initially extracted, the elimination process resulted in larger subexpressions. The process works as follows. We extract common subexpressions by finding pairs of cubes, making them cube-free, and choosing the ones that appear most often. This selected subset is evaluated more precisely by performing exact semi-algebraic division as described above. Binary and MV variables are treated uniformly.

7 Conclusions

We have given several semi-algebraic methods for multi-valued logic functions. The methods treat binary and multi-valued variables uniformly. They include methods for

- finding common sub-expressions,
- exact and inexact semi-algebraic division,
- factoring an expression, and
- decomposing a multi-valued network.

The algorithms have been implemented in a system being developed, MV-SIS, where they were tested and tuned for quality on a small set of examples. MV-SIS also includes methods for simplifying an expression (such as extraction of network don't cares and use of ESPRESSO-MV) as well as most other operations found in SIS, but extended to the MV case. Our initial implementations show that the new algebraic methods are fast and effectively fill the need for methods faster than Boolean methods.

Even though all common subexpressions extracted by the methods of this paper are MV-functions (binary output), multi-valued output functions can be created, after extraction, by pairing one or more MV-functions and treating the combination as a single multi-valued output function. Groups of pairwise orthogonal MV-functions are equivalent to a single multi-valued output function. We can optionally put more weight on extracting functions that are orthogonal to already extracted functions or choosing to extract, at one time, a set of mutually orthogonal functions. Groups of functions which are not mutually orthogonal can be interpreted as a single multi-valued output function whose output values have been binary encoded. Grouping several binary functions together creates essentially a multi-output PLA which can be minimized by Espresso-MV. However, finding the best way to group functions remains an area for more research. This problem has elements in common with input encoding, output encoding, phase assignment and bit pairing problems.

Acknowledgements

This work was supported by the SRC under contract 683.004 and through the California Micro program by Fujitsu, Synopsys, and Cadence.

References

[1] R. K. Brayton, M. Chiodo, R. Hojati, T. Kam, K. Kodandapani, R. P. Kurshan, S. Malik, A. L. Sangiovanni-Vincentelli, E. M. Sentovich, T. Shiple, K. J. Singh, and H.-Y. Wang. BLIF-MV: An Interchange Format for Design Verification and Synthesis. Technical Report UCB/ERL M91/97, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, November 1991.

[2] R. K. Brayton and C. McMullen. The Decomposition and Factorization of Boolean Expressions. In *Proc. of the Intl. Symposium on Circuits and Systems*, pages 49–54, May 1982.

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

[4] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. SIS: A System for Sequential Circuit Synthesis. Technical Report UCB/ERL M92/41, Electronics Research Laboratory, Univ. of California, Berkeley, CA 94720, May 1992.

[5] J. Vasudevamurthy and J. Rajski. A Method for Concurrent Decomposition and Factorization of Boolean Expressions. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 510–513, November 1990.

Appendix: Direct Method for Exact Division

Problem 1 *Given a set of cubes F and an divisor $\{K_i, 1 \leq i \leq n\}$, find the largest quotient $\{J_i, 1 \leq i \leq m\}$.*

We give a method for finding a solution for $n = 2$ and generalize it later to $n > 2$.

1. Find all candidate cubes for K_1 and K_2: any cube $C_k \subseteq K_i$ is a candidate cube for K_i:

$K_1 : C_{11}, C_{12}, ..., C_{1n_1}, C_{1i} \subseteq K_1, 1 \leq i \leq n_1$

$K_2 : C_{21}, C_{22}, ..., C_{2n_2}, C_{2i} \subseteq K_2, 1 \leq i \leq n_2$

There may be common cubes for K_1 and K_2. In this case, we can duplicate the cubes in $F : C + C = C$. This is analogous to the binary case: $x + x = x$.

2. For each candidate cube C_{ij}, compute a candidate cokernel cube Q_{ij} that has a lower and an upper bound on the values it can take. For each literal k,

$$C_{ij}^k \subseteq Q_{ij}^k \subseteq (C_{ij} + \tilde{K}_i)^k \qquad (1)$$

Q_{ij} is a cokernel for K_i and C_{ij} *iff* (1) is satisfied for each literal k.

Express Q_{ij} in positional notation where we denote Q_{ij}^{kl} as the value l of variable k in cube Q_{ij}:

$$Q_{ij}^{kl} = \begin{cases} 1 & \text{if } C_{ij}^{kl} = 1 \\ 0 & \text{if } (C_{ij} + \tilde{K}_i)^{kl} = 0 \\ 2 & \text{otherwise} \end{cases}$$

3. The cokernel cubes for K_1 and for K_2 form a bipartition B_1 and B_2. We want to find the maximum compatability of it. We define $J = \emptyset$ if either $Q_1^{kl} = 1$ and $Q_2^{kl} = 0$ or $Q_1^{kl} = 0$ and $Q_2^{kl} = 1$ for any k, l. A cokernel cube Q_1 from B_1 is compatible with a cube Q_2 from B_2 *iff* $J \neq \emptyset$. If $J \neq \emptyset$, we compute $J = Q_1 \star Q_2$ such that:

$$J^{kl} = Q_1^{kl} \star Q_2^{kl} = 1 \Longleftrightarrow \begin{cases} Q_1^{kl} = 1, Q_2^{kl} = 1 \\ Q_1^{kl} = 2, Q_2^{kl} = 1 \\ Q_1^{kl} = 1, Q_2^{kl} = 2 \end{cases}$$

$$J^{kl} = Q_1^{kl} \star Q_2^{kl} = 0 \Longleftrightarrow \begin{cases} Q_1^{kl} = 0, Q_2^{kl} = 2 \\ Q_1^{kl} = 2, Q_2^{kl} = 0 \\ Q_1^{kl} = 2, Q_2^{kl} = 2 \end{cases}$$

Example:

	X_1	X_2	X_3
Q_1	1111	1020	1102
Q_2	1111	1000	1102
J	1111	1000	1100

So $J = X_2^{\{0\}} X_3^{\{0,1\}}$ and this is the desired cokernel cube. We want $J^{kl} = 0$ if $Q_1^{kl} = Q_2^{kl} = 2$ in order to minimize the number of literal values in the selected cokernel cube.

4. For $n > 2$, the solution is the same except for the $\star$ operation, which generalizes to finding the maximum compatible n cokernel cubes from n partitions. It is equivalent to finding a maximum clique within the values of each literal. Here we use the fact that if n cubes interesect pairwise, then there is a single cube common to all n cubes. By finding all bipartite cliques of size n, and marking all cliques that have no points in common (the cliques are independent), we can solve a maximum independent set problem to find the best matching.

Example 12 *Consider:*

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

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

where $c^{\{3\}} + a^{\{0\}}c^{\{0,1,2\}}$ is the given divisor. Then Candidate cubes:
$c^{\{3\}} : a^{\{0,1,2\}}c^{\{3\}}, b^{\{1,2,3\}}c^{\{3\}}$
$a^{\{0\}}c^{\{0,1,2\}} : a^{\{0\}}c^{\{1\}}, a^{\{0\}}b^{\{1,2,3\}}c^{\{0\}}$
Corresponding cokernel cubes:
$c^{\{3\}} : a^{\{0,1,2\}}c^{S_1}, b^{\{1,2,3\}}c^{S_2}$
$a^{\{0\}}c^{\{0,1,2\}} : a^{S_3}c^{S_4}, a^{S_5}b^{\{1,2,3\}}c^{S_6}$
where $\{3\} \subseteq S_1 \subseteq \{0,1,2,3\}, \{3\} \subseteq S_2 \subseteq \{0,1,2,3\}, \{0\} \subseteq S_3 \subseteq \{0,1,2,3\}, \{1\} \subseteq S_4 \subseteq \{1,3\}, \{0\} \subseteq S_5 \subseteq \{0,1,2,3\}, \{0\} \subseteq S_6 \subseteq \{0,3\}$.
Take the cokernel cubes in common (e.g., make S_1 and S_4 compatible), we get:
$c^{\{3\}} : a^{\{0,1,2\}}c^{\{1,3\}}, b^{\{1,2,3\}}c^{\{0,3\}}$
$a^{\{0\}}c^{\{0,1,2\}} : a^{\{0,1,2\}}c^{\{1,3\}}, b^{\{1,2,3\}}c^{\{0,3\}}$
So the cokernel cubes obtained are: $a^{\{0,1,2\}}c^{\{1,3\}}$ and $b^{\{1,2,3\}}c^{\{0,3\}}$. This is what we would get by using the satisfiability matrix method.