Minimization of Multiple-Valued Functions in Post Algebra

Elena Dubrova
Department of Electronics
Royal Institute of Technology
Stockholm, Sweden
elena@ele.kth.se

Yunjian Jiang Robert Brayton
Dept. of Electrical Engineering and Computer Sciences
University of California
Berkeley, USA
{wjiang,brayton}@eecs.berkeley.edu

Abstract

The relation between minimum sum-of-products expressions for multiple-valued input, binary-valued output functions and minimum sum-of-products expressions in Post algebra for multiple-valued input, multiple-valued output functions is made precise. We give an algorithm for minimizing Post expressions and discuss a method for their factorization. Experiments are given to demonstrate the effectiveness of the algorithm.

1 Introduction

Multiple-valued logic is a generalization of classical Boolean logic. At higher levels of abstraction, where the variables often range over a set of symbolic values, use of multiple-valued logic can make the design task more intuitive [2]. For example, a traffic light controller with a signal *light* $\in$ {*red, yellow, green*} is conceptually easier than encoding $light_1\overline{light}_0 = green$ etc. The designer can first manipulate and optimize a multiple-valued network, and then, when no further optimization is possible in the multiple-valued domain, perform a suitable encoding and derive a Boolean network. This allows a better exploration of the design space, since the binary encoding is postponed and multiple-valued optimization is not influenced by such decisions. At the final stage, several binary encodings might be tried to select a good one.

Multiple-valued functions were studied originally to provide support for designing m-valued logic circuits, which employ m discrete signals, $m \geq 2$. Attempts to build multiple-valued integrated circuits (ICs) can be traced back to 1970, starting from early work on 3-valued designs [15]. Applying multiple-valued logic can enhance circuit performance in terms of chip area, operation speed and power consumption [1], [6]. Multiple-valued circuits have matured to the point where four-valued flash and DRAM memories are commercially available [13].

Techniques for manipulating multiple-valued functions have been used to design binary-valued circuits that implement some encoding of the multiple values. Examples are synthesis of PLAs with input encoders [16], optimization of finite state machines [8], identification of Boolean factors [7].

The best way to benefit from multiple-valued logic is to manipulate a multiple-valued representation directly. When no further optimization is possible, a suitable binary encoding can be applied. This approach is not used due to the lack of mature packages for representation, manipulation, synthesis, optimization and verification of multiple-valued functions. To gain popularity, these packages must be competitive with other efficient logic packages, like CUDD [18] and VIS [19]. Such multiple-valued packages are not developed yet mainly because a complete suite of associated algorithms is still not available. Although some techniques are generalized to the multiple-valued case, many essential ones are either missing or inefficient.

In this paper we address two fundamental problems in multiple-valued theory: minimization and factorization. We clarify the relation between minimum sum-of-products expressions for multiple-valued input, binary-valued output functions and minimum sum-of-products expressions in Post algebra for multiple-valued input, multiple-valued output functions. We give an algorithm for minimizing Post expressions and discuss a method for their factorization. Experiments are given to demonstrate the effectiveness of the algorithm.

2 Multiple-valued functions

A multiple-valued function is a discrete function whose input and output variables take two or more values. Formally, an n-variable multiple-valued function $f(x_1, \ldots, x_n)$ is a mapping $f : P_1 \times P_2 \times \ldots \times P_n \rightarrow$

M, with the variables x_i taking values from the sets $P_i = \{0, 1, 2, \ldots, p_i - 1\}$, $p_i > 1$, $i \in \{1, 2, \ldots, n\}$, and the function f taking its value from the set $M = \{0, 1, 2, \ldots, m-1\}$, $m > 1$.

As an algebraic system for manipulation of multiple-valued functions focus on *chain based Post algebra*, corresponding to the first multiple-valued logic developed by Emil Post in 1921 [12]. This is based on a totally ordered set of elements and the operations maximum (MAX), minimum (MIN) and literal. It is a special case of more general Post algebras, based on lattices [11]. In our case, the set is the union of the input domains of the variables and the output domain of the function, i.e $M \cup (\cup_{i=1}^{n} P_i)$. *Maximum* $\text{MAX}(x_i, x_j)$, $i, j \in \{1, 2, \ldots, n\}$, is a binary operation of type $P_i \times P_j \rightarrow M$, whose output is the larger of the two values of its arguments. *Minimum* $\text{MIN}(x_i, x_j)$ outputs the smaller of the two values. The *literal* x_i^S is a unary operation of type $P_i \rightarrow M$, defined by

$$x_i^S = \begin{cases} m-1 & \text{if } x \in S \\ 0 & \text{otherwise} \end{cases}$$

where $x_i \in P_i$ is a multiple-valued variable and $S \subseteq P_i$ is a set. Throughout the paper, we omit brackets when S is a single-element set, i.e. we write x_i^j instead of $x_i^{\{j\}}$.

Cube is a product of literals of type $c \cdot x_1^{S_1} \cdot x_2^{S_2} \cdot \ldots \cdot x_n^{S_n}$, where $c \in M$ is a constant and product "$\cdot$" is MIN. A *sum-of-products* expression over Post algebra is a sum of cubes, with sum being MAX. For example, the function shown in Figure 1 can be written as:

$$f(x_1, x_2) = 1 \cdot x_1^1 \cdot x_2^{\{0,2\}} + 2 \cdot x_2^1$$

with "+" = MAX and "$\cdot$" = MIN.

$x_2 \backslash x_1$	0	1	2
0	0	1	0
1	2	2	2
2	0	1	0

Figure 1: An example of 3-valued function.

Since we use the MAX operation between cubes, a cube with a higher-valued coefficient may be combined with a cube with a lower-valued coefficient as shown in the following Lemma.

Lemma 1 *Let a and b be constants in M and g, h, k be multiple-valued literals. If $f = a \cdot g + b \cdot h$ and $a < b$, then $f = a \cdot k + b \cdot h$ where $g \subseteq k \subseteq (g \cup h)$.*

Using Lemma 1, a sum-of-product expression for the function in Figure 1 can be simplified to

$$f(x_1, x_2) = f(x_1, x_2) = 1 \cdot x_1^1 + 2 \cdot x_2^1$$

We can also omit the constant $m - 1$ from product-terms, since $m - 1 \cdot a = a$ for any $a \in M$.

To generalize the notions of on-set and off-set for the multiple-valued case, we first extend the operation literal over a variable to the operation literal over the function:

$$f^i(x_1, \ldots, x_n) = \begin{cases} m-1 & \text{if } f(x_1, \ldots, x_n) = i \\ 0 & \text{otherwise} \end{cases}$$

where $i \in M$ is a constant. The literal f^i contains the minterms mapped to i by $f(x_1, \ldots, x_n)$. Since $f^i \cdot f^{\{j\}} = 0$ for any $i \neq j, i, j \in M$, the input domain $P_1 \times P_2 \times \ldots \times P_n$ of f can be partitioned into m disjoint sets as:

$$f(x_1, \ldots, x_n) = \sum_{i=0}^{m-1} i \cdot f^i(x_1, \ldots, x_n) \qquad (1)$$

We use the same notation f^i to denote the set $\{x | f(x) = i\}$ and call this the *i-set* of $f(x_1, \ldots, x_n)$. In the two-valued case, *0-set* corresponds to the off-set and *1-set* corresponds to the on-set of the function.

3 Minimization

3.1 Completely specified functions

Two-level sum-of-products for multiple-valued functions and associated minimization techniques have been studied since 1970 [9], [10], [17]. The algorithms for minimization of Boolean sum-of-products expressions, beginning with the generation of all prime implicants followed by the selection of a minimum cover, can be extended to multiple-valued expressions over Post algebra, but it is quite inefficient. A more efficient way, which we follow in this paper, is to partition the function with respect to each of its non-zero values $i \in M - \{0\}$ into i-sets, and then apply an efficient minimizer for multiple-valued input, binary-valued output functions, like Espresso-MV [14], to find a minimum cover for each of i-sets. These are called *priority* don't cares. More formally, such an algorithm is given as follows.

Algorithm 1: *For $f : P_1 \times P_2 \times \ldots \times P_n \rightarrow M$:*

1. Compute i-sets f^i for all $i \in \{1, 2, \ldots, m-1\}$.

2. For each $i = \{m-1, m-2, \ldots, 1\}$, find a minimum cover F_i^{min} for the multiple-valued input, binary-valued output function (F_f, D_f, R_f) with the on-set

$F_f = f^i$, priority don't care set $D_f = \cup_{j=i+1}^{m-1} f^{\{j\}}$ and off-set $R_f = \cup_{j=0}^{i-1} f^{\{j\}}$.

3. The cover F for f is given by $F = \cup_{i=1}^{m-1} i \cdot F_i^{min}$.

In Step 2, the on-set contains all the cubes mapped to $m-1$ by f^i, the priority don't care set contains all the cubes mapped to $m-1$ by $f^{\{j\}}$, for all j greater than i, and the off-set contains all the cubes mapped to $m-1$ by $f^{\{j\}}$, for all j smaller than i, $i, j \in M$. Multiple-valued input, binary-valued output functions are minimized successively for $i = m-1, m-2, \ldots, 1$. Finally, Equation 1 is applied to merge the minimized covers into a cover for the original function. Such an algorithm for minimization of multiple-valued functions was used for the first time in [3]. However, no formal proof was given to assure that the algorithm really computes the minimal cover for the function. Below we present a formal statement and proof.

Theorem 1 *Algorithm 1 finds a minimum cover for a multiple-valued function $f : P_1 \times P_2 \times \ldots \times P_n \rightarrow M$.*

Proof: Suppose F is not a minimum cover for f. Then there exists another cover, F', and an index $i \in M$ such that $|F_i'| < |F_i^{min}|$. From Lemma 1, $f^i \subseteq F_i' \subseteq (f^i \cup (\cup_{j=i+1}^{m-1} f^j))$. On the other hand, from Step 2 of Algorithm 1, $f^i \subseteq F_i^{min} \subseteq (f^i \cup (\cup_{j=i+1}^{m-1} f^j))$ because the cover F_i^{min} is found using all $f_j, j > i$, as don't cares. Since F_i^{min} is minimum, $|F_i'| = |F_i^{min}|$, which contradicts the above assumption. Thus, $|F| = |F'|$ and F is a minimum cover for f.

□

As an example, consider the 3-valued 2-variable function $f : \{0,1,2\}^2 \rightarrow \{0,1,2\}$ shown in Figure 1. Its i-sets for $i = 1, 2$ are:

$$f^1 = x_1^1 x_2^0 + x_1^1 x_2^2$$
$$f^2 = x_1^0 x_2^1 + x_1^1 x_2^1 + x_1^2 x_2^1$$

First, we simplify f^2 to x_2^1 and then minimize f^1 using x_2^1 as a don't care set. This results in $f^1 = x_1^1$. Finally, we merge these two functions to get:

$$f = 1 \cdot f^1 + f^2 = 1 \cdot x_1^1 + x_2^1$$

3.2 Incompletely specified functions

In this section we show that the problem of minimization of incompletely specified functions is equivalent to the problem of minimization of completely specified functions with one more value in the output domain.

Consider the following algorithm:

Algorithm 2: *For $f : P_1 \times P_2 \times \ldots \times P_n \rightarrow M \cup \{-\}$:*

1. Replace all don't care values in the output domain with the value m.

2. Compute i-sets f^i for all $i \in \{1, 2, \ldots, m\}$.

2. For each $i = \{m-1, m-2, \ldots, 1\}$, find a minimum cover F_i^{min} for the multiple-valued input, binary-valued output function (F_f, D_f, R_f) with the on-set $F_f = f^i$, priority don't care set $D_f = \cup_{j=i+1}^{m} f^{\{j\}}$ and off-set $R_f = \cup_{j=0}^{i-1} f^{\{j\}}$.

3. The cover F for f is given by $F = \cup_{i=1}^{m-1} i \cdot F_i^{min}$.

Note, that the cover F_m^{min}, corresponding to the don't care value, is not included in the resulting cover for the function.

Next, we state that Algorithm 2 computes a minimum cover for the function. The proof is similar to that of Theorem 1.

Theorem 2 *Algorithm 2 finds a minimum cover for an incompletely specified multiple-valued function $f : P_1 \times P_2 \times \ldots \times P_n \rightarrow M \cup \{-\}$.*

3.3 Output phase optimization

Sometimes, a better result can be obtained by renaming the values in the output domain of the function to obtain a different ordering. For example, consider the function shown in Figure 2 (left). A minimum sum-of-products expression over Post algebra is

$$f = 1 \cdot x_1^3 + 1 \cdot x_2^0 + x_1^{\{0,1,2\}} x_2^1 + x_1^2 x_2^{\{1,2,3\}} + 2 \cdot x_1^0 x_2^3.$$

If we rename the values in the output domain as shown in Figure 2 (right), a minimum sum-of-products form is

$$f = 1 \cdot x_1^{\{0,1,2\}} x_2^{\{1,2,3\}} + 2 \cdot x_1^{\{0,1\}} x_2^{\{2,3\}} + x_1^3 x_2^3.$$

In general, an output ordering problem needs to be solved to get a minimum sum-of-products expression for a multiple-valued function. This problem is an analog of the output phase assignment problem in the Boolean case.

4 Factoring

If we have a minimum sum-of-products expression over Post algebra, we may often simplify it further by factoring. We can look for a (algebraic) factorization of the type $F = D \cdot Q + R$ where $\cdot$ represents the MIN operation and $+$, the MAX operation. This leads

$x_2 \backslash x_1$	0	1	2	3
0	1	1	1	1
1	3	3	3	1
2	0	0	3	1
3	2	0	3	1

$x_2 \backslash x_1$	0	1	2	3
0	0	0	0	0
1	1	1	1	0
2	2	2	1	0
3	3	2	1	0

Figure 2: Re-ordering values in output domain.

to a slight extension of a factoring operation defined for multiple-valued input, binary-valued output functions [4]. Note that a product of two expressions can be written as an array, e.g.

$$\begin{aligned}(3\cdot x^3+2\cdot x^2+1\cdot x^1)\cdot(3\cdot y^3+2\cdot y^2+1\cdot y^1)\\ =\begin{array}{ccc} 3\cdot x^3\cdot y^3 & 2\cdot x^3\cdot y^2 & 1\cdot x^3\cdot y^1\\ 2\cdot x^2\cdot y^3 & 2\cdot x^2\cdot y^2 & 1\cdot x^2\cdot y^1\\ 1\cdot x^1\cdot y^3 & 1\cdot x^1\cdot y^2 & 1\cdot x^1\cdot y^1\end{array}\end{aligned}$$

Thus, to find $D \cdot Q$ we need to find a subset of the cubes of F that can be arranged in a "satisfiable" matrix (which is the same as defined in [4], except that now we have constants to deal with). The constants in the matrix must be able to be written as an outer product (MIN operation) of two constant vectors. As an example, the satisfiable matrix

$$\begin{array}{ccc} 3\cdot x^{\{0,3\}}\cdot u^1 & 2\cdot x^{\{0,1\}}\cdot v^1 & 2\cdot x^{\{0,1,3\}}\cdot w^1\\ 1\cdot x^{\{0,2,3\}}\cdot y^1\cdot u^1 & 1\cdot x^{\{0,1,2\}}\cdot y^1\cdot v^1 & 1\cdot y^1\cdot w^1\\ 3\cdot x^{\{0,2,3\}}\cdot z^1\cdot u^1 & 2\cdot x^{\{0,1,2\}}\cdot z^1\cdot v^1 & 2\cdot z^1\cdot w^1\end{array}$$

provides a factorization $(3\cdot x^{\{0,1,3\}}+1\cdot y^1+3\cdot z^1)\cdot$ $\cdot(3\cdot x^{\{0,2,3\}}\cdot u^1+2\cdot x^{\{0,1,2}\cdot v^1+2\cdot w^1)$.

Given a satisfiable matrix, the factorization is obtained by doing row and column operations, where for each row, we take the maximum coefficient, and the supercube of the row entries; similarly for the columns. Then one factor is the sum of the row results, the other is the sum of the column results. Recall, that if a variable is not present in a cube, the literal with all values is there. Note also that, given a satisfiable matrix, the two factors are not necessarily unique.

Duality of MIN and MAX operations can be exploited to look for a "factorization" of the type f MAX g.

5 Experimental results

We have implemented the minimization Algorithm 1 in MVSIS [5] and applied it to a set of benchmark functions. Most of the examples are from the multi-valued logic benchmark suite available from Portland State University [20]; some are hand-made examples. The characteristics of the benchmarks are summarized in Table1. It shows, for each example, the number of inputs (PI) followed in parenthesis by the maximum range of their values, and the number of outputs (PO) followed by the maximum range of their values.

Table 1: MV-network examples

example	PI	PO	example	PI	PO
adder-mod4	3 (4)	1(4)	nursery	8 (3-5)	1 (5)
decoder	3 (4)	2(8)	balance	4 (5)	1 (5)
plus8	2 (8)	1(15)	car	6 (3-4)	1 (4)
pal3x	6 (3)	1(4)	mm3	5 (3)	1 (3)
matmul	8 (3)	4(3)	mm4	5 (4)	1 (4)
xor-mux	3 (4-6)	1(4)	mm5	5 (5)	1 (5)
employ1	9 (3-5)	1(4)	iris	4 (5-12)	1 (3)
employ2	7 (3-5)	1(4)	chess1	6 (4-8)	1(18)
max	8 (8)	1(8)			

The results of minimization are summarized in Table 2. Column *orig* shows, after multi-level examples are collapsed into two-level sum-of-product expressions over Post algebra, the number of multi-valued cubes in the multiple-valued functions when each i-set is minimized without priority don't cares. The rest of the columns show the number of cubes obtained using two different heuristics for output phase assignment. The first heuristic (column *heur-1*) assigns to each i-set a priority. The highest priority i-set is associated with $m-1$, etc. Consider the following merit function, which is the inverse of the weighted sum of the cube count C_i and the literal count L_i for the cover, i.e.

$$priority_i = \frac{1}{\alpha C_i + \beta L_i}.$$

The reasoning is that: (1) high priority i-sets have smaller representations and do not need and cannot use don't cares as effectively; (2) functions with few literals, may have more minterms and hence produce more priority don't cares for downstream i-sets.

Note that the i-set with the largest representation is given the lowest priority, so that when it is minimized we always obtain the empty set. We tried heuristics for $\{\alpha,\beta\} = \{0,1\}$, $\{1,0\}$ and $\{10,1\}$. The experiments show that, among these, the best ordering was provided by either $\{1,0\}$ or $\{0,1\}$, but never by $\{10,1\}$. In the discussion that follows we select the best result between $\{1,0\}$ and $\{0,1\}$ for each heuristic.

Column *heur-2* shows the results of a second heuristic, which is the reverse of *heur-1*, i.e. the order of the i-sets is in reverse priority, except that the largest i-set is always kept as the default. Column *rand* is a random ordering but again with the largest i-set being the default. On average, there is a 25% reduction when the first heuristic is used with the minimization algorithm. The first heuristic is on average better than its reverse, but there are a few cases when the random ordering

Table 2: Minimization results (# cubes)

example	orig	heur-1	heur-2	rand
adder-mod4	48	36	36	28
decoder	14	14	14	14
plus8	56	44	56	39
pal3x	138	94	122	90
matmul	96	88	80	80
xor-mux	48	28	28	28
employ1	19	19	19	19
employ2	36	21	31	21
nursery	48	32	44	44
balance	104	68	86	101
car	32	28	32	28
mm3	10	10	10	10
mm4	34	23	30	27
mm5	79	69	52	53
iris	29	28	28	28
max	49	7	49	42
chess1	2651	1770	1746	1757
average ratio	1	0.75	0.87	0.80

gives the best result (*adder-mod4, plus8, pal3x*). Another way to look at the results is that *heur-1* gives a result, equal to the best, in 11 out of 17 times; *heur-2* in 7 out of 17 times; and *rand* in 11 out of 17 times. Taking the best of the three columns gives an average ratio of 0.72. This suggests a need for a better heuristic, which we will look into in our future experiments.

6 Conclusion

We showed how to find a minimum two-level sum-of-products expression over chain-based Post algebra for incompletely specified multiple-valued functions. We gave an algorithm for minimizing Post expressions, using techniques for minimization of binary-output multi-valued functions. Experimental results showed about 25% reduction in the size of the cover compared to an algorithm which minimizes each i-set separately. We noted that output phase assignment can improve the results.

References

[1] Ben Dhaou I., Dubrova E., Tenhunen H. 2001 Power efficient inter-module communication for digit-serial DSP architectures in deep-submicron technology *Proc. 31st Int. Symp. Multiple-Valued Logic*

[2] Brayton R. K., Khatri S. P. 1999 Multi-valued logic synthesis *Proc. 12th Int. Conf. on VLSI Design* 196-206.

[3] Dueck G. W., Miller D. M., 1988 Direct search minimization of multiple-valued functions *Proc. 18th Int. Symp. Multiple-Valued Logic* 218-25

[4] Gao M. and Brayton R., 2000 Semi-Algebraic Methods for Multi-Valued Logic *Proceedings of the International Workshop on Logic Synthesis*

[5] M. Gao, J.-H. Jiang, Y. Jiang, Y. Li, S. Sinha, R. Brayton, 2001 MVSIS, *submitted to IWLS'01*

[6] Hanyu T., Kameyama M. 1995 A 200 MHz pipelined multiplier using 1.5 V-supply multiple-valued MOS current-mode circuits with dual-rail source-coupled logic, *IEEE Journal of Solid-State Circuits* **30**(11) 1239-45

[7] Liao S., Devadas S., Ghosh A. 1993 Boolean factorization using multiple-valued minimization *Proc. Int. Conf. Computer-Aided Design* 606-11

[8] De Micheli G. D., Brayton R., Sangiovanni-Vincentelli A. 1984 KISS: a program of optimal state encoding of finite state machines *Proc. Int. Conf. Computer-Aided Design*

[9] Moraga C. 1971 A minimization method for 3-valued logic functions in *Theory of Machines and Computations* (New York: Academic) 363-75

[10] Muzio J. C. and Miller, D. M. 1979 On the minimization of many-valued functions *Proc. 9th Int. Symp. Multiple-Valued Logic* 294-9

[11] Muzio J. C. and Wesselkamper T. C. 1986 *Multiple-valued switching theory* (Bristol: Hilger)

[12] Post E. L. 1921 Introduction to a general theory of elementary propositions *Amer. J. Math.* **43** 163-85

[13] B. Ricco et al. 1998 Non-volatile multilevel memories for digital applications *Proc. IEEE* **86**(12) 2399-2421

[14] Rudell R. and Sangiovanni-Vincentelli A. 1987 Multiple-valued minimization for PLA optimization *IEEE Trans. om Computer-Aided Design* **5** 727-50

[15] Smith K. C. 1981 The prospects for multivalued logic: A technology and applications view, *IEEE Trans. om Computers* **C-30**(9) 619-34

[16] Sasao T. Multiple-valued logic and optimization of programmable logic arrays 1988 *IEEE Computer* **21** 71-80

[17] Smith W. R. 1997 Minimization of multiple-valued functions in *Computer science and multiple-valued logic* ed: Rine (North-Holland:Amsterdam)

[18] Somenzi F. 1998 *CUDD: CU Decision Diagram Package, Release 2.3.0* University of Colorado at Boulder

[19] The VIS Group, VIS: A system for Verification and Synthesis 1996 *Proc. 8th Int. Conf. on Computer Aided Verification* Springer Lecture Notes in Computer Science ed: Alur and Henzinger **1102**

[20] Portland Logic Optimization group, Portland State University, http://www.ee.pdx.edu/ polo/