Journal of Electronic Testing (2019) 35:679–694
https://doi.org/10.1007/s10836-019-05831-x

Equivalence Checking and Compaction of n-input Majority Terms Using Implicants of Majority

Rajeswari Devadoss[1] · **Kolin Paul**[1] · **M. Balakrishnan**[1]

Received: 18 May 2019 / Accepted: 11 September 2019 / Published online: 30 November 2019

Abstract

Recent advances in nanotechnology have led to the emergence of energy efficient circuit technologies like spintronics and domain wall magnets that use Majority gates as their primary logic elements. For logic synthesis methods targeted to such technologies to be effective and efficient, they need to be able to use, manipulate, and exploit large Majority terms in their synthesis flow. One of the problems that turn up in such an endeavor is the determination of equivalence of two n-input Majority terms. As Majority gates implement more complex Boolean functions than traditional AND/OR gates, this is a non-trivial problem—one that demands to be solved before proceeding to harder problems dealing with networks of Majority gates. We provide an algorithm based on prime implicants as a solution to this problem. In addition, we provide an algorithm that compacts an n-input Majority term to an equivalent n-input Majority term that has the fewest number of inputs. In this quest, we extend the concept of implicants to two cases — 1-implicants and prime 1-implicants that imply that a function evaluates to '1', and 0-implicants and prime 0-implicants that imply that it evaluates to '0'. We exploit the properties of Majority to create an efficient algorithm to generate the sums of all prime 1-implicants and all prime 0-implicants of an n-input Majority term. As Majority and Threshold functions have been shown to be logically equivalent, our algorithms are applicable to Threshold functions as well. Being based on implicants of Majority, our algorithms improve on the known naive algorithms for equivalence checking and compaction for threshold logic terms.

Keywords Majority · Threshold · Prime implicants · Implicants · Canonical form · Equivalence checking · Compaction · Minimization

1 Introduction

Spintronics is an emerging technology where devices work by manipulating spin rather than charge. It offers the 3-input Majority as the primary logic element which produces an output '1' if at least 2 of its inputs are '1's, and produces an output '0' otherwise [6, 8]. As it can be converted into a 2-input AND (or OR) gate by setting one of its inputs to the constant '1' (or '0'), it packs more computational power than 2-input AND/OR gates while forming a functionally complete set with the NOT gate. Not only are there more upcoming fabrics based on 3-input Majority gates, some, like Quantum-dot Cellular Automata (QCA) may even offer Majority gates of more than three inputs [1]. It is established that the 3-input Majority gate can be used to implement Boolean functions in compact forms [9].

As the promise of building efficient logic circuits increases by leaps with Majority gates coming into the picture, so does the problem of creating logic synthesis methods that can exploit them. The existing repository of logic synthesis techniques based on and for AND/OR logic, while readily available, under-perform when targeted to Majority logic. It has become clear that the best way to build logic synthesis techniques for Majority is to use generic n-input terms to represent and manipulate Boolean expressions [2]. This leads us to the turf of having to frequently determine the equivalence of two n-input

Responsible Editor: M. Chen

✉ Rajeswari Devadoss
rajidrc@gmail.com

Kolin Paul
kolin@cse.iitd.ac.in

M. Balakrishnan
mbala@cse.iitd.ac.in

[1] Department of Computer Science and Engineering, IIT Delhi, New Delhi 110016, India

Majority terms. While it is trivial to check the equivalence of two AND (or OR) terms, the same cannot be said of n-input Majority terms. This becomes clear once we formally define an n-input Majority operation and observe it in action.

Definition 1 (**Majority Operation**) When n is odd, the Majority operation on a **multiset** W_n **of** n **Boolean elements** results in '1' if there are more '1's than '0's in it, and in '0' otherwise. That is, $\langle W_n \rangle$=1 if at least $\frac{n+1}{2}$ elements of the multiset W_n are '1's, and $\langle W_n \rangle$=0 otherwise. *The number of times an element m occurs in a multiset* W_n *is called its multiplicity l, and is written as* m^l. The cardinality n of a multiset W_n is the total number of elements in it, including duplicates, and is marked as its subscript. A Majority operation on a multiset is called a "Majority term", and the multiset its "underlying multiset".

The fact that the Majority operation applies to a *multiset* that might contain elements with different multiplicities makes determining equivalence of two Majority terms non-trivial. For instance, all of $\langle a^5\, b^4\, c^2 \rangle$, $\langle a^3\, b^3\, c^3 \rangle$, $\langle a^2\, b^2\, c^2\, d \rangle$, and the compact term $\langle a\, b\, c \rangle$ are logically equivalent and represent the function $ab+bc+ca$. Table 1 shows 18 Majority terms that are all equivalent to the compact term $\langle a\ b\ c\ d\ e\ f^2\ g^2 \rangle$. Not only is their equivalence not "trivially" evident (as with AND/OR terms), it is hard to see that they implement the Boolean function shown. It is even harder to *compact* any of them to $\langle a\ b\ c\ d\ e\ f^2\ g^2 \rangle$. As n grows in W_n, so do the number of possible combinations of multiplicities of its elements, and more complicated the determination of equivalence and compaction.

As expected from a barely studied subject as the mathematics of n-input Majority operations, little is known about matters such as equivalence checking that is not naive to the algebra of Majority terms. The intuitive solution to this problem seems to be to use existing tools that have been optimized to check the equivalence of not one term, but whole logic networks. This approach brings up a fundamental problem—translating an n-input Majority term into AND/OR terms so that traditional tools understand the Boolean function that it implements.

Thus, we have three problems that need solutions informed by an awareness of Majority behavior:

- Generate a minimal representation of an n-input Majority term that can be understood by traditional AND/OR based tools
- Efficiently check for the equivalence of two n-input Majority terms without translating them to other representations
- "Compact" a given n-input Majority term to an equivalent term with the least possible cardinality

Being fundamental to the understanding of a *single n-input Majority term*, these problems need to be solved to be able to proceed to bigger problems involving large Majority networks. Hence, we focus on the single Majority term and not address Majority networks in this paper.

In this paper, we study the mathematical behavior of the n-input Majority operation and apply the insights derived to solve these problems. We create an efficient algorithm that checks for the logical equivalence of two n-input Majority terms without handing the problem over to traditional Majority-ignorant tools. For this purpose, we extend the concept of implicants by introducing the 0-implicant as a dual to the traditional (1-)implicant. We exploit symmetry in Majority to compactly represent 1,0-implicants and prime 1,0-implicants of n-input Majority terms. We utilize the recursive property of these implicants to provide an efficient algorithm that is capable of producing ***two canonical representations of a given n-input Majority term - sums of prime 1,0-implicants***. With implicants as the basis, we apply duality to prune this process and use it to check for the equivalence of two n-input Majority terms efficiently. In addition, we provide an efficient algorithm to **compact** a given n-input Majority term to minimize the cardinality of its underlying multiset while keeping its Boolean function the same.

Having provided a glimpse of the n-input Majority term Equivalence Checking algorithm and the idea of implicants

Table 1 Multiple Majority Terms implementing the shown Boolean function

COMPACT TERM : $\langle a\ b\ c\ d\ e\ f^2\ g^2 \rangle$		
$\langle a^3\ b^3\ c^3\ d^3\ e^3\ f^6\ g^8 \rangle$	$\langle a^3\ b^3\ c^3\ d^3\ e^3\ f^5\ g^5 \rangle$	$\langle w\ a^3\ b^3\ c^3\ d^3\ e^3\ f^6\ g^7 \rangle$
$\langle a^2\ b^3\ c^3\ d^3\ e^4\ f^7\ g^7 \rangle$	$\langle a^2\ b^3\ c^3\ d^3\ e^3\ f^6\ g^7 \rangle$	$\langle w\ a^2\ b^3\ c^3\ d^3\ e^3\ f^6\ g^6 \rangle$
$\langle a^3\ b^3\ c^3\ d^3\ e^3\ f^7\ g^7 \rangle$	$\langle a^3\ b^3\ c^3\ d^3\ e^3\ f^5\ g^7 \rangle$	$\langle w\ a^3\ b^3\ c^3\ d^3\ e^3\ f^5\ g^6 \rangle$
$\langle a^3\ b^3\ c^3\ d^3\ e^4\ f^6\ g^7 \rangle$	$\langle a^2\ b^3\ c^3\ d^3\ e^4\ f^6\ g^6 \rangle$	$\langle w\ a^3\ b^3\ c^3\ d^3\ e^4\ f^6\ g^6 \rangle$
$\langle a^3\ b^3\ c^3\ d^3\ e^5\ f^6\ g^6 \rangle$	$\langle a^3\ b^3\ c^3\ d^3\ e^3\ f^6\ g^6 \rangle$	$\langle w\ a^2\ b^2\ c^2\ d^2\ e^2\ f^4\ g^4 \rangle$
$\langle a^3\ b^3\ c^3\ d^4\ e^4\ f^6\ g^6 \rangle$	$\langle a^3\ b^3\ c^3\ d^3\ e^4\ f^5\ g^6 \rangle$	$\langle w^2\ a^3\ b^3\ c^3\ d^3\ e^3\ f^6\ g^6 \rangle$

$fg(a+b+c+d+e)+(f+g)(abc+abd+abe+acd+ace+ade+bcd+bce+bde+cde)+abcde$

for Majority in our paper in VLSI Design 2019, we detail the underlying mathematical proofs and show how our methods compares with the existing method. We also present an n-input Majority term Compaction algorithm based on similar ideas [3].

2 Related Work

It is well-established that Majority gates and Threshold gates are logically equivalent and interchangeable [7]. Hence, the mathematical properties and algorithms that apply to Majority logic networks apply to Threshold logic networks, and vice versa. However, in this work we deal only with individual Majority/Threshold functions by considering their inputs as literals and not logic networks.

To our knowledge, the only work that has contributions towards presenting the Boolean function implemented by a Threshold function is by Gowda et. al [4]. This work presents an algorithm that produces the Maximally Factored form of a given threshold function, and uses the result in combination with a Boolean Expression Diagram (BED) based tool to check the equivalence of Threshold circuits [5]. Instead of using a Maximally Factored form that can produce very large expressions for Majority gates with large number of inputs, we use a compact count-based form. Since we are concerned with only a single Majority gate and not logic networks, the algorithm that we provide is much more efficient and tuned to the properties of a single Majority/Threshold gate. For instance, Gowda et. al. would need to generate the Maximally Factored forms of two given Threshold gates and run BED based equivalence checking on the results. Our equivalence checking method applies the mathematics of Majority and performs equivalence checking without generating the Boolean expressions of the functions. This is helpful in Majority (Threshold) logic synthesis where there is a necessity to quickly check the equivalence of two Majority/Threshold terms.

On the other hand, early exploration of Majority and Threshold logic has led to various studies of constructing a minimized representation of a given Threshold/Majority function. To our knowledge, the best culmination of these studies is an algorithm that appears in the book "Threshold logic and its applications" by Saburo Muroga (1971). The Algorithm 6.4.1.1 (page 175 of the book) proceeds to attempt to minimise the weights of a self-dual positive unate Threshold function by building an inequality representing each prime implicant of the function and solving the resulting linear programming problem [7]. This, or any similar algorithm in Majority/Threshold logic literature fails to exploit the mathematics of Majority to reduce the complexity of the problem being solved. Our method applies the properties of implicants of Majority terms to greatly reduce the search space for a compact representation.

3 Implicants and Majority

The canonical form — the computational signature of a Boolean function — helps us understand the conditions under which it evaluates to '1' and to easily verify the equivalence of Boolean functions. The **Blake canonical form**, the sum of all prime implicants of a Boolean function is one of the more interesting canonical forms of any Boolean function. An implicant in this representation of a Boolean function is a term that implies the function — that is, the function evaluates to '1' whenever the implicant does. Another way to look at implication is to know when the truth of a term implies that the function evaluates to '0'. These two approaches are in fact duals in the context of canonical forms. We extend the concept of implicant to two cases — the 1-implicant which implies that the function evaluates to '1', and the 0-implicant which implies that the function evaluates to '0'.

Definition 2 (**Implicants**) A product term T is a 1-implicant of a Boolean function F if T implies F, that is, if F evaluates to '1' whenever T evaluates to '1'. Similarly, a product term T is a 0-implicant of a Boolean function F if T implies $\overline{F}$, that is, if F evaluates to '0' whenever T evaluates to '1'.

Similarly, we extend the concept of a prime implicant to two cases — prime 1-implicant and prime 0-implicant.

Definition 3 (**Prime Implicants**) A prime 1-implicant of a Boolean function F is a 1-implicant that cannot be covered by a more general 1-implicant. Similarly, a prime 0-implicant of a Boolean function F is a 0-implicant that cannot be covered by a more general 0-implicant. A prime 0-implicant of F is a prime 1-implicant of $\overline{F}$, and vice-versa.

Not only is the sum of all prime 1-implicants a canonical representation of Boolean functions, the negation of the sum of all prime 0-implicants is one too.

As we know, the Majority operation is based on counting the number of '1's and '0's among its inputs. This means that, the implicants of a Majority operation too can be constructed using counts of '1's and '0's in its multiset - that is, using count-terms of the form $X_{\geq k}$ (mincount-term) and $X_{\leq l}$ (maxcount-term) that represent "at least k '1's in X" and "at most l '1's in X" respectively.

Definition 4 (**Count Terms**) Given a multiset X_m, the Mincount-term $X_{\geq k}$ on it evaluates to '1' if and only if **at least** k elements of X_m are '1's. Similarly, the Maxcount-term $X_{\leq k}$ on it evaluates to '1' if and only if **at most** k elements of X_m are '1's. They can be implemented by Majority terms of the form $\langle X_m\, c^l \rangle$ with the appropriate number of copies of a constant c as shown below:

$$X_{\geq k} = \begin{cases} 1 & \text{if } k \leq 0 \\ \langle X_m 1^{m+1-2k} \rangle & \text{if } 0 < k \leq \frac{m+1}{2} \\ \langle X_m 0^{2k-m-1} \rangle & \text{if } \frac{m+1}{2} \leq k \leq m \\ 0 & \text{if } k > m \end{cases}$$

$$X_{\leq k} = \begin{cases} 0 & \text{if } k < 0 \\ \langle \overline{X}_m 0^{m-1-2k} \rangle & \text{if } 0 \leq k \leq \frac{m-1}{2} \\ \langle \overline{X}_m 1^{2k-m+1} \rangle & \text{if } \frac{m-1}{\leq} k < m \\ 1 & \text{if } k \geq m \end{cases}$$

We propose that products of mincount-terms (or maxcount-terms) of subsets of the elements with the same multiplicity in a Majority term are ideal for the representation of its implicants and prime-implicants.

Let us define a ***Scenario A*** for use in our considerations of implicants in this work: Function $F = \langle {}^{1-}W_{n_1}^{l_1}\, {}^{2-}W_{n_2}^{l_2} \cdots {}^{m-}W_{n_m}^{l_m} 1^p 0^q \rangle$ is defined on the variables in the set $V = \{ {}^{1-}W_{n_1}, {}^{2-}W_{n_2}, \cdots, {}^{m-}W_{n_m} \}$ where each ${}^{i-}W_{n_i}$, called a Symmetry-group, is the set of variables $\{w_{i_1}, w_{i_2}, \ldots, w_{i_{n_i}}\}$; and a subset of these variable sets $\{ {}^{k_1-}W_{n_{k1}}, {}^{k_2-}W_{n_{k2}}, \cdots, {}^{k_s-}W_{n_{ks}} \} \subseteq V$. This function F is a positive unate Majority/Threshold function, and its properties can be applied to all Majority/Threshold functions after transforming them into positive unate Majority terms.

The products of mincount-terms (or maxcount-terms) on the underlying multiset of F are easily recognized as its implicants by checking the conditions specified in Theorems 1, 2, 3 and 4

Theorem 1 (**1-implicant of Majority**) *Consider a function F as described by **Scenario A**. The product term ${}^{k_1-}W_{\geq c_{k_1}}\, {}^{k_2-}W_{\geq c_{k_2}} \cdots {}^{k_s-}W_{\geq c_{k_s}}$ is a 1-implicant of F if and only if F evaluates to '1' when exactly $c_{k_1}, c_{k_2}, \cdots, c_{k_s}$ variables of the sets ${}^{k_1-}W_{n_{k1}}, {}^{k_2-}W_{n_{k2}}, \ldots, {}^{k_s-}W_{n_{ks}}$ respectively are '1's. That is, the product of mincount-terms ${}^{k_1-}W_{\geq c_{k_1}}\, {}^{k_2-}W_{\geq c_{k_2}} \cdots {}^{k_s-}W_{\geq c_{k_s}}$ is a 1-implicant of F if and only if its truth assures at least the number of '1's that the Majority-term implementing F needs to evaluate to '1'. This condition is written as*

$$c_{k_1} l_{k_1} + \cdots + c_{k_s} l_{k_s} + p \geq \frac{n_1 l_1 + n_2 l_2 + \cdots + n_m l_m + p + q + 1}{2}$$

Proof The product term ${}^{k_1-}W_{\geq c_{k_1}}\, {}^{k_2-}W_{\geq c_{k_2}} \cdots {}^{k_s-}W_{\geq c_{k_s}}$ evaluates to '1' if and only if at least $c_{k_1}, c_{k_2}, \cdots, c_{k_s}$ variables of the sets ${}^{k_1-}W_{n_{k1}}, {}^{k_2-}W_{n_{k2}}, \ldots, {}^{k_s-}W_{n_{ks}}$ respectively are '1's.

This term can imply the function F only if its base case of exactly $c_{k_1}, c_{k_2}, \cdots, c_{k_s}$ variables of the sets ${}^{k_1-}W_{n_{k1}}, {}^{k_2-}W_{n_{k2}}, \ldots, {}^{k_s-}W_{n_{ks}}$ respectively being '1's provides the sufficient number of '1's for the Majority-term $\langle {}^{1-}W_{n_1}^{l_1}\, {}^{2-}W_{n_2}^{l_2} \cdots {}^{m-}W_{n_m}^{l_m} 1^p 0^q \rangle$ to evaluate to '1'. That is

$$c_{k_1} l_{k_1} + c_{k_2} l_{k_2} + \cdots + c_{k_s} l_{k_s} + p \geq \frac{n_1 l_1 + n_2 l_2 + \cdots + n_m l_m + p + q + 1}{2}$$

If this condition is true, the similar condition

$$d_{k_1} l_{k_1} + d_{k_2} l_{k_2} + \cdots + d_{k_s} l_{k_s} + p \geq \frac{n_1 l_1 + n_2 l_2 + \cdots + n_m l_m + p + q + 1}{2}$$

holds true as long as each d_{k_i} is greater than or equal to the corresponding c_{k_i}. Hence, the function F evaluates to '1' if the product ${}^{k_1-}W_{\geq c_{k_1}}\, {}^{k_2-}W_{\geq c_{k_2}} \cdots {}^{k_s-}W_{\geq c_{k_s}}$ evaluates to '1'. Thus proved. □

Theorem 2 (**0-implicant of Majority**) *Consider a function F as described by **Scenario A**. The product term ${}^{k_1-}W_{\leq c_{k_1}}\, {}^{k_2-}W_{\leq c_{k_2}} \cdots {}^{k_s-}W_{\leq c_{k_s}}$ is a 0-implicant of F if and only if F evaluates to '0' when exactly $c_{k_1}, c_{k_2}, \cdots, c_{k_s}$ variables of the sets ${}^{k_1-}W_{n_{k1}}, {}^{k_2-}W_{n_{k2}}, \ldots, {}^{k_s-}W_{n_{ks}}$ respectively are '1's. That is, the product of maxcount-terms ${}^{k_1-}W_{\leq c_{k_1}}\, {}^{k_2-}W_{\leq c_{k_2}} \cdots {}^{k_s-}W_{\leq c_{k_s}}$ is a 0-implicant of F if and only if its truth assures at least the number of '0's that the Majority-term implementing F needs to evaluate to '0'. This condition is written as*

$$(n_{k_1} - c_{k_1}) l_{k_1} + (n_{k_2} - c_{k_2}) l_{k_2} + \cdots + (n_{k_s} - c_{k_s}) l_{k_s} + q \geq \frac{l_1 n_1 + l_2 n_2 + \cdots + l_m n_m + p + q + 1}{2}$$

Proof The product term ${}^{k_1-}W_{\leq c_{k_1}}\, {}^{k_2-}W_{\leq c_{k_2}} \cdots {}^{k_s-}W_{\leq c_{k_s}}$ evaluates to '1' if and only if at most $c_{k_1}, c_{k_2}, \cdots, c_{k_s}$ variables of the sets ${}^{k_1-}W_{n_{k1}}, {}^{k_2-}W_{n_{k2}}, \ldots, {}^{k_s-}W_{n_{ks}}$ respectively are '1's.

This term can imply the function F only if its base case of exactly $c_{k_1}, c_{k_2}, \cdots, c_{k_s}$ variables of the sets ${}^{k_1-}W_{n_{k1}}, {}^{k_2-}W_{n_{k2}}, \ldots, {}^{k_s-}W_{n_{ks}}$ respectively being '1's provides the sufficient number of '0's for the Majority-term $\langle {}^{1-}W_{n_1}^{l_1}\, {}^{2-}W_{n_2}^{l_2} \cdots {}^{m-}W_{n_m}^{l_m} 1^p 0^q \rangle$ to evaluate to '1'. As the number of '0's is the difference between the number of variables and the number of '1's in a set, this condition is:

$$(n_{k_1} - c_{k_1}) l_{k_1} + (n_{k_2} - c_{k_2}) l_{k_2} + \cdots + (n_{k_s} - c_{k_s}) l_{k_s} + q \geq \frac{l_1 n_1 + l_2 n_2 + \cdots + l_m n_m + p + q + 1}{2}$$

If this condition is true, the similar condition

$$(n_{k_1}-d_{k_1})l_{k_1}+(n_{k_2}-d_{k_2})l_{k_2}+\cdots+(n_{k_s}-d_{k_s})l_{k_s}+q \geq \frac{l_1n_1+l_2n_2+\cdots+l_mn_m+p+q+1}{2}$$

holds true as long as each d_{k_i} is less than or equal to the corresponding c_{k_i}. Hence, the function F evaluates to '0' if the product term ${}^{k_1-}W_{\leq c_{k_1}}\ {}^{k_2-}W_{\leq c_{k_2}}\cdots{}^{k_s-}W_{\leq c_{k_s}}$ evaluates to '1'. Thus proved. □

Theorem 3 *Consider a function F as described by* ***Scenario A*** *with l_{k_1} as the smallest of the multiplicities l_{k_i}.*

The product term ${}^{k_1-}W_{\geq c_{k_1}}\ {}^{k_2-}W_{\geq c_{k_2}}\cdots{}^{k_s-}W_{\geq c_{k_s}}$ is a prime 1-implicant of F if and only if it implies F but the product term ${}^{k_1-}W_{\geq (c_{k_1}-1)}\ {}^{k_2-}W_{\geq c_{k_2}}\ {}^{k_3-}W_{\geq c_{k_3}}\cdots{}^{k_s-}W_{\geq c_{k_s}}$ ***does*** *not.*

Proof By Definition 3, $T={}^{k_1-}W_{\geq c_{k_1}}\ {}^{k_2-}W_{\geq c_{k_2}}\cdots{}^{k_s-}W_{\geq c_{k_s}}$ is a prime 1-implicant of F only if it is a 1-implicant of F. Also, it is a prime 1-implicant only if it is not covered by a more general 1-implicant. Any term that covers the term T is of the form ${}^{k_1-}W_{\geq d_{k_1}}\ {}^{k_2-}W_{\geq d_{k_2}}\cdots{}^{k_s-}W_{\geq d_{k_s}}$ where each $d_i \leq c_i$. Such a term ensures $(c_{k_1}-d_{k_1})l_{k_1}+(c_{k_2}-d_{k_2})l_{k_2}+\cdots+(c_{k_s}-d_{k_s})l_{k_s}$ fewer '1's than the term T does. Of these, the term $T'={}^{k_1-}W_{\geq (c_{k_1}-1)}\ {}^{k_2-}W_{\geq c_{k_2}}\ {}^{k_3-}W_{\geq c_{k_3}}\cdots{}^{k_s-}W_{\geq c_{k_s}}$ ensures l_{k_1} fewer '1's than the term T does. Since l_{k_1} is the smallest of the multiplicities l_{k_i}, any other term of the considered form ensures even fewer '1's, and cannot imply the function F unless T' does. Hence, the term T is a prime 1-implicant of the function F only if the term T' is not.

Conversely, if the term T implies F but T' does not, it means that no other term that covers T implies F either. Hence, the term T is a prime 1-implicant of the function F in this case. □

Theorem 4 *Consider a function F as described by* ***Scenario A*** *with l_{k_1} as the smallest of the multiplicities l_{k_i}.*

The product term ${}^{k_1-}W_{\leq c_{k_1}}\ {}^{k_2-}W_{\leq c_{k_2}}\cdots{}^{k_s-}W_{\leq c_{k_s}}$ is a prime 0-implicant of F if and only if it implies $\overline{F}$ but the product term ${}^{k_1-}W_{\leq (c_{k_1}+1)}\ {}^{k_2-}W_{\leq c_{k_2}}\ {}^{k_3-}W_{\leq c_{k_3}}\cdots{}^{k_s-}W_{\leq c_{k_s}}$ ***does*** *not.*

Proof By Definition 3, $T={}^{k_1-}W_{\leq c_{k_1}}\ {}^{k_2-}W_{\leq c_{k_2}}\cdots{}^{k_s-}W_{\leq c_{k_s}}$ is a prime 0-implicant of F only if it is a 0-implicant of F. Also, it is a prime 0-implicant only if it is not covered by a more general 0-implicant. Any term that covers the term T is of the form ${}^{k_1-}W_{\leq d_{k_1}}\ {}^{k_2-}W_{\leq d_{k_2}}\cdots{}^{k_s-}W_{\leq d_{k_s}}$ where each $d_i \geq c_i$. Such a term ensures $(d_{k_1}-c_{k_1})l_{k_1}+(d_{k_2}-c_{k_2})l_{k_2}+\cdots+(d_{k_s}-c_{k_s})l_{k_s}$ more '1's than the term T does. Of these, the term $T'={}^{k_1-}W_{\leq (c_{k_1}+1)}\ {}^{k_2-}W_{\leq c_{k_2}}\ {}^{k_3-}W_{\leq c_{k_3}}\cdots{}^{k_s-}W_{\leq c_{k_s}}$ ensures l_{k_1} more '1's than the term T does. That is, it ensures l_{k_1} fewer '0's than the term T does. Since l_{k_1} is the smallest of the multiplicities l_{k_i}, any other term of the considered form ensures even fewer '0's, and cannot imply the function $\overline{F}$ unless T' does. Hence, the term T is a prime 0-implicant of the function F only if the term T' is not.

Conversely, if the term T implies $\overline{F}$ but T' does not, it means that no other term that covers T implies $\overline{F}$ either. Hence, the term T is a prime 0-implicant of the function F in this case. □

4 Sum of all Prime Implicants of a Majority term

Having found a way to recognize prime implicants of a Majority term, we now proceed to the task of generating the canonical form of a Majority term — the sum of all of its prime 1-implicants (or prime 0-implicants). Rather than a brute force or iterative examination of all possible implicants of a Majority term, we opt for a recursive method. We use the observation that a product term including a mincount-term ${}^{i-}W_{\geq k}$ implies a Majority term if and only if its cofactor with respect to ${}^{i-}W_{\geq k}$ implies the Majority term where k and $n_i - k$ elements of ${}^{i-}W_{n_i}$ are substituted with '1' and '0' respectively. This can be used as the basis of recursion in implicants of a Majority term as shown in Lemma 1 and Theorem 5.

Lemma 1 (**Recursion in Implicants**) *Consider a function F as described by* ***Scenario A****. The product term ${}^{k_1-}W_{\geq c_{k_1}}\ {}^{k_2-}W_{\geq c_{k_2}}\cdots{}^{k_s-}W_{\geq c_{k_s}}$ is a 1-implicant of F if and only if the product term ${}^{k_1-}W_{\geq c_{k_1}}\ {}^{k_2-}W_{\geq c_{k_2}}\cdots{}^{k_{(s-1)}-}W_{\geq c_{k_{(s-1)}}}$ is a 1-implicant of the function G implemented by the Majority-term*

$$\langle {}^{1-}W^{l_1}_{n_1}\ {}^{2-}W^{l_2}_{n_2}\cdots{}^{(k_s-1)-}W^{l_{(k_s-1)}}_{n_{(k_s-1)}}\ {}^{(k_s+1)-}W^{l_{(k_s+1)}}_{n_{(k_s+1)}}\ {}^{(K_s+2)-}W^{l_{(k_s+2)}}_{n_{(k_s+2)}}\cdots{}^{m-}W^{l}_{n}\ 1^{p+c_{k_s}l_{k_s}}0^{q+(n_{k_s}-c_{k_s})l_{k_s}}\rangle$$

Similarly, the product term ${}^{k_1-}W_{\leq c_{k_1}}\ {}^{k_2-}W_{\leq c_{k_2}}\cdots{}^{k_s-}W_{\leq c_{k_s}}$ is a 0-implicant of F if and only if the product term ${}^{k_1-}W_{\leq c_{k_1}}\ {}^{k_2-}W_{\leq c_{k_2}}\cdots{}^{k_{(s-1)}-}W_{\leq c_{k_{(s-1)}}}$ is a 0-implicant of the function G.

Proof Both the functions F and G are implemented by Majority-terms on multisets of the same cardinality, and

hence need the same lim number of '1's (or '0's) to evaluate to '1' (or '0').

According to Theorem 1, the product term ${}^{k_1-}W_{\geq c_{k_1}}\ {}^{k_2-}W_{\geq c_{k_2}} \cdots {}^{k_s-}W_{\geq c_{k_s}}$ is a 1-implicant of F if and only if

$$c_{k_1}l_{k_1}+c_{k_2}l_{k_2}+\cdots+c_{k_{(s-1)}}l_{k_{(s-1)}}+c_{k_s}l_{k_s}+p \geq lim$$

Also, the term ${}^{k_1-}W_{\geq c_{k_1}}\ {}^{k_1-}W_{\geq c_{k_2}} \cdots {}^{k_{(s-1)}-}W_{\geq c_{k_{(s-1)}}}$ is a 1-implicant of G if and only if

$$c_{k_1}l_{k_1}+c_{k_2}l_{k_2}+\cdots+c_{k_{(s-1)}}l_{k_{(s-1)}}+p+c_{k_s}l_{k_s} \geq lim$$

These two conditions are identical. Hence, the term ${}^{k_1-}W_{\geq c_{k_1}}\ {}^{k_2-}W_{\geq c_{k_2}} \cdots {}^{k_s-}W_{\geq c_{k_s}}$ is a 1-implicant of F if and only if the product ${}^{k_1-}W_{\geq c_{k_1}}\ {}^{k_2-}W_{\geq c_{k_2}} \cdots {}^{k_{(s-1)}-}W_{\geq c_{k_{(s-1)}}}$ is a 1-implicant of G.

Similarly, according to Theorem 2, the term ${}^{k_1-}W_{\leq c_{k_1}}\ {}^{k_2-}W_{\leq c_{k_2}} \cdots {}^{k_s-}W_{\leq c_{k_s}}$ is a 0-implicant of F if and only if

$$(n_{k_1}-c_{k_1})l_{k_1}+(n_{k_2}-c_{k_2})l_{k_2}+\cdots+(n_{k_{(s-1)}}-c_{k_{(s-1)}})l_{k_{(s-1)}} +(n_{k_s}-c_{k_s})l_{k_s}+q \geq lim$$

Also, the term ${}^{k_1-}W_{\leq c_{k_1}}\ {}^{k_2-}W_{\leq c_{k_2}} \cdots {}^{k_{(s-1)}-}W_{\leq c_{k_{(s-1)}}}$ is a 0-implicant of the function G if and only if

$$(n_{k_1}-c_{k_1})l_{k_1}+(n_{k_2}-c_{k_2})l_{k_2}+\cdots+(n_{k_{(s-1)}}-c_{k_{(s-1)}})l_{k_{(s-1)}} +q+(n_{k_s}-c_{k_s})l_{k_s} \geq lim$$

These conditions are identical. Hence, the product ${}^{k_1-}W_{\leq c_{k_1}}\ {}^{k_2-}W_{\leq c_{k_2}} \cdots {}^{k_s-}W_{\leq c_{k_s}}$ is a 0-implicant of F if and only if the product ${}^{k_1-}W_{\leq c_{k_1}}\ {}^{k_2-}W_{\leq c_{k_2}} \cdots {}^{k_s-}W_{\leq c_{k_s}}$ is a 0-implicant of G. □

Theorem 5 *Consider a function F as per* ***Scenario A*** *with l_{k_1} as the smallest of the multiplicities l_{k_i}. The product term ${}^{k_1-}W_{\geq c_{k_1}}\ {}^{k_2-}W_{\geq c_{k_2}} \cdots {}^{k_s-}W_{\geq c_{k_s}}$ is a prime 1-implicant of F if and only if the product term ${}^{k_1-}W_{\geq c_{k_1}}\ {}^{k_2-}W_{\geq c_{k_2}} \cdots {}^{k_{(s-1)}-}W_{\geq c_{k_{(s-1)}}}$ is a prime 1-implicant of the cofactor function G implemented by the Majority-term*

$$\langle {}^{1-}W_{n_1}^{l_1}\ {}^{2-}W_{n_2}^{l_2} \cdots {}^{(k_s-1)-}W_{n_{(k_s-1)}}^{l_{(k_s-1)}}\ {}^{(k_s+1)-}W_{n_{(k_s+1)}}^{l_{(k_s+1)}}\ {}^{(k_s+2)-}W_{n_{(k_s+2)}}^{l_{(k_s+2)}} \cdots {}^{m-}W_{n}^{l}\ 1^{p+c_{k_s}l_{k_s}}0^{q+(n_{k_s}-c_{k_s})l_{k_s}}\rangle$$

Similarly, the product term ${}^{k_1-}W_{\leq c_{k_1}}\ {}^{k_2-}W_{\leq c_{k_2}} \cdots {}^{k_s-}W_{\leq c_{k_s}}$ is a prime 0-implicant of F if and only if the product term ${}^{k_1-}W_{\leq c_{k_1}}\ {}^{k_2-}W_{\leq c_{k_2}} \cdots {}^{k_{(s-1)}-}W_{\leq c_{k_{(s-1)}}}$ is a prime 0-implicant of the function G.

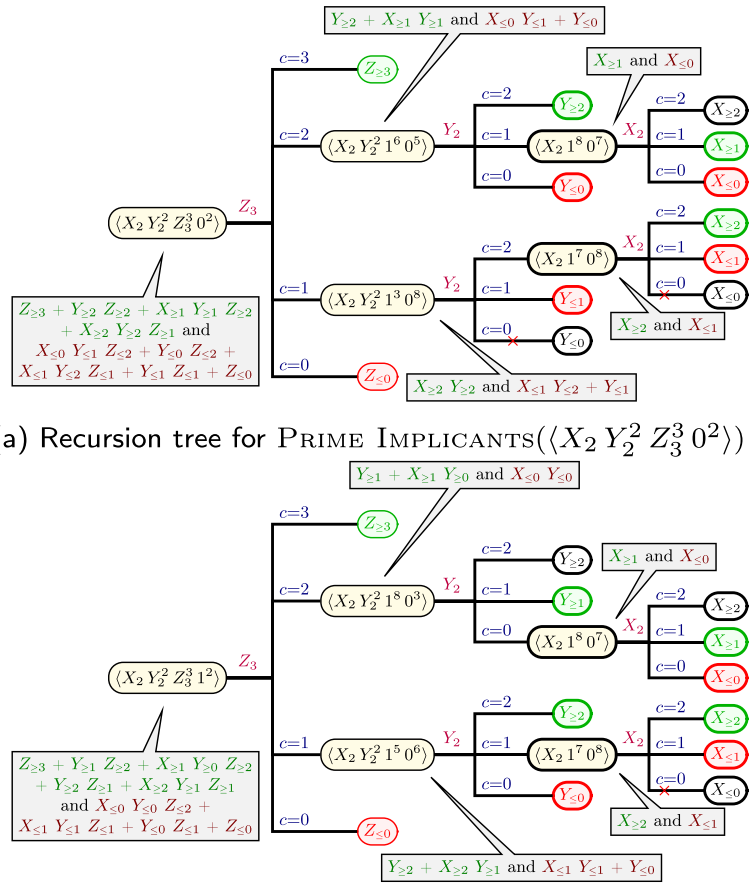
Proof As per Theorem 3, the product term $T={}^{k_1-}W_{\geq c_{k_1}}\ {}^{k_2-}W_{\geq c_{k_2}} \cdots {}^{k_s-}W_{\geq c_{k_s}}$ is a prime 1-implicant of F if and only if it is a 1-implicant of the function F but the term $T'={}^{k_1-}W_{\geq (c_{k_1}-1)}\ {}^{k_2-}W_{\geq c_{k_2}}\ {}^{k_3-}W_{\geq c_{k_3}} \cdots {}^{k_s-}W_{\geq c_{k_s}}$ is not.

Also, $S={}^{k_1-}W_{\geq c_{k_1}}\ {}^{k_2-}W_{\geq c_{k_2}} \cdots {}^{k_{(s-1)}-}W_{\geq c_{k_{(s-1)}}}$ is a prime 1-implicant of G if and only if it is a 1-implicant of G, but $S'={}^{k_1-}W_{\geq (c_{k_1}-1)}\ {}^{k_2-}W_{\geq c_{k_2}}\ {}^{k_3-}W_{\geq c_{k_3}} \cdots {}^{k_{(s-1)}-}W_{\geq c_{k_{(s-1)}}}$ is not. Additionally, according to Lemma 1, the term T is a 1-implicant of F if and only if the term S is a 1-implicant of G. Also, the term T' is not a 1-implicant of F if and only if the term S' is not a 1-implicant of G. Hence, the term T is a prime 1-implicant of F if and only if the term S is a prime 1-implicant of G.

Similarly, as per Theorem 4, the term T is a prime 0-implicant of the F if and only if it is a 0-implicant of F but the term $T''={}^{k_1-}W_{\geq (c_{k_1}+1)}\ {}^{k_2-}W_{\geq c_{k_2}}\ {}^{k_3-}W_{\geq c_{k_3}} \cdots {}^{k_s-}W_{\geq c_{k_s}}$ is not. Also, the term S is a prime 0-implicant of the G if and only if it is a 0-implicant of G but the term $S''={}^{k_1-}W_{\geq (c_{k_1}+1)}\ {}^{k_2-}W_{\geq c_{k_2}}\ {}^{k_3-}W_{\geq c_{k_3}} \cdots {}^{k_{(s-1)}-}W_{\geq c_{k_{(s-1)}}}$ is not. Additionally, as per Lemma 1, the term T is a 0-implicant of F if and only if the term S is a 0-implicant of G. Also, the term T'' is not a 0-implicant of F if and only if the term S'' is not a 0-implicant of G. Hence, the term T is a prime 0-implicant of F if and only if the term S is a prime 0-implicant of G. □

We now use this knowledge to construct a recursive method that generates the sum of all prime 1-implicants and the sum of all prime 0-implicants of a given Majority term. This function shown in Algorithm 1 builds the prime implicants of a Majority term from the bottom up. It requires the Majority term to be in the form $\langle {}^{1-}W_{n_1}^{l_1}\ {}^{2-}W_{n_2}^{l_2} \cdots {}^{m-}W_{n_m}^{l_m}1^p0^q\rangle$ with the multiplicities occurring in ascending order $l_i \leq l_{i+1}$. It begins with a single count-term on the set of variables with the highest multiplicity $W_{n_m}^{l_m}$. Then, it includes more count-terms to this product term till it becomes a prime implicant. It uses the recursive nature of prime implicants to generate these additional count-terms.

Note that if the term ${}^{k_1-}W_{\geq c_{k_1}}\ {}^{k_2-}W_{\geq c_{k_2}} \cdots {}^{k_s-}W_{\geq c_{k_s}}$ is not a 1-implicant of a function f, then neither is ${}^{k_1-}W_{\geq (c_{k_1}-1)}\ {}^{k_2-}W_{\geq c_{k_2}}\ {}^{k_3-}W_{\geq c_{k_3}} \cdots {}^{k_s-}W_{\geq c_{k_s}}$, and it need not be checked for being one. Similarly, if the term ${}^{k_1-}W_{\leq c_{k_1}}\ {}^{k_2-}W_{\leq c_{k_2}} \cdots {}^{k_s-}W_{\leq c_{k_s}}$ is a 0-implicant of a function f, then so is ${}^{k_1-}W_{\leq (c_{k_1}-1)}\ {}^{k_2-}W_{\leq c_{k_2}}\ {}^{k_3-}W_{\leq c_{k_3}} \cdots {}^{k_s-}W_{\leq c_{k_s}}$, and it need not be checked for being one.



(a) Recursion tree for PRIME IMPLICANTS($\langle X_2\, Y_2^2\, Z_3^3\, 0^2 \rangle$)

(b) Recursion tree for PRIME IMPLICANTS($\langle X_2\, Y_2^2\, Z_3^3\, 1^2 \rangle$)

Fig. 1 The sum of all prime 1-implicants and sum of all prime 0-implicants of the functions f and g implemented by the Majority terms $\langle X_2 Y_2^2 Z_3^3 0^2 \rangle$ and $\langle X_2 Y_2^2 Z_3^3 1^2 \rangle$ respectively can be generated recursively using Algorithm 1

In Fig. 1, we illustrate the workings of this function on the two Majority terms $\langle X_2 Y_2^2 Z_3^3 0^2 \rangle$ and $\langle X_2 Y_2^2 Z_3^3 1^2 \rangle$. These terms contain three groups of symmetric variables X_2, Y_2 and Z_3 with multiplicities one, two and three respectively. Hence, the recursion trees generating the prime implicants of these terms grow to three levels - one level each for the three groups (X, Y, Z) in descending order of their multiplicities. In each level of these trees, a prime 1-implicant is represented in green and a prime 0-implicant is represented in red.

At each level, the function checks if any the mincount-term on the group of variables with the largest multiplicity is a prime 1-implicant of the term being processed. For example, in the first level of Fig. 1a, $Z_{\geq 3}$ is a prime 1-implicant while $Z_{\geq 2}$ is not. Next, it checks if any maxcount-term on the group of variables with the largest multiplicity is a prime 0-implicant of the term being processed. For example, in the first level of Fig. 1a, $Z_{\leq 0}$ is a prime 0-implicant while $Z_{\leq 1}$ and $Z_{\leq 2}$ are not. For a given count k, it makes a recursive call to generate the prime implicants only when neither case is true. Hence, in the first level of Fig. 1a, recursive calls are made for the cases that two and one elements of Z_3 are '1's.

Algorithm 1 All Prime 1-implicants and Prime 0-implicants of a Majority term.

Require: Multiplicities in ascending order: $1_i \leq l_{i+1}$ for all i
Require: The term does not have an absolute Majority of '1' or '0'
function PRIME IMPLICANTS($\langle {}^{1-}W_{n_1}^{l_1}\ {}^{2-}W_{n_2}^{l_2} \cdots {}^{m-}W_{n_m}^{l_m} 1^p 0^q \rangle$)
  $primeOne \leftarrow 0$ ▷ Sum of all prime 1-implicants of the term
  $primeZero \leftarrow 0$ ▷ Sum of all prime 0-implicants of the term
  $lim \leftarrow \frac{n_1 l_1 + n_2 l_2 + \cdots + n_m l_m + p + q + 1}{2}$ ▷ Number needed for Majority
  $c \leftarrow n_m$
  while $c \geq 0$ and $cl_m + p \geq lim$ **do** ▷ Skip 1-implicant (Th.1)
    $c \leftarrow c - 1$
  end while
  $primeOne \leftarrow primeOne + {}^{m-}W_{\geq c+1}$ ▷ Prime 1-implicant (Th.3)
  while $c \geq 0$ and $(n_m - c)l_m + q < lim$ **do** ▷ Recurse (Th.5)
    $subTerm \leftarrow \langle {}^{1-}W_{n_1}^{l_1}\ {}^{2-}W_{n_2}^{l_2} \cdots {}^{m-1}W_{n_{m-1}}^{l_{m-1}} 1^{p+cl_m} 0^{q+(n_m-c)l_m} \rangle$
    $subPrimeOne, subPrimeZero \leftarrow$ PRIME IMPLICANTS($subTerm$)
    $primeOne \leftarrow primeOne + subPrimeOne.\ {}^{m-}W_{\geq c}$
    $primeZero \leftarrow primeZero + subPrimeZero.\ {}^{m-}W_{\leq c}$
    $c \leftarrow c - 1$
  end while
  if $c \geq 0$ **then** ▷ Prime 0-implicant (Th.4)
    $primeZero \leftarrow primeZero + {}^{m-}W_{\leq c}$
  end if
  return $primeOne, primeZero$
end function

When it makes a recursive call for count k on ${}^{i-}W_{n_i}$, the function multiplies the sum of prime 1-implicants that it receives by ${}^{i-}W_{\geq k}$ and adds it to the sum of all of its prime

1-implicants. Similarly, it multiplies the sum of prime 0-implicants that it receives by ${}^{i-}W_{\leq k}$ and adds it to the sum of all of its prime 0-implicants. For instance, the call for the node for $c = 2$ in the first level of Fig. 1a returns the sums $Y_{\geq 2} + X_{\geq 1} Y_{\geq 1}$ and $X_{\leq 0} Y_{\leq 1} + Y_{\leq 0}$. The former is multiplied by $Z_{\geq 2}$ and added to the sum of prime 1-implicants while the latter is multiplied by $Z_{\geq 2}$ and added to the sum of prime 0-implicants. The function returns the sums of all prime 1-implicants and prime 0-implicants generated across counts.

In this figure, the nodes that are implicants but not prime implicants are uncolored. Also, the 0-implicants that are not prime 0-implicants are never visited by the algorithm. Hence, these nodes representing 0-implicants are marked by a red cross.

Note that this technique provides a canonical representation of the given Majority term when no two variables from different Symmetry-groups are symmetric to each other. When this is not the case, the technique produces a correct and compact representation, though not canonical.

Thus, we have a method to generate a compact representation of the sum of all prime 1-implicants and the sum of all prime 0-implicants of a Majority term.

5 Equivalence of Majority Terms

The foremost use of a canonical representation of Boolean functions is that it reduces the task of checking the equivalence of two functions to transforming them to their canonical forms. Such a canonical form is crucial to the Majority operation since multiple Majority terms on a set of variables often implement the same function. For instance, all 3-input Majority terms $\langle a^p\, b^q\, c^r \rangle$ that do not have a variable with absolute Majority over the remaining variables implement the function $ab + bc + ca$. The fact that no variable has absolute Majority means that the other two variables together contribute more '1's or '0's than it. Hence, the term evaluates to '1' whenever two of its variables are '1's, and implements the same function as $\langle a\, b\, c \rangle$.

The sum of all prime 1-implicants (or all prime 0-implicants) generated using Algorithm 1 is a canonical form that can be used to detect equivalence of two Majority terms on the same set of variables with the same symmetry grouping. For example, as shown in Fig. 2, the sum of all prime 1-implicants (or all prime 0-implicants) of the Majority term $\langle X_2\, Y_2^3\, Z_3^4\, 0^3 \rangle$ is the same as that of $\langle X_2 Y_2^2 Z_3^3 0^2 \rangle$ shown in Fig. 1a.

In fact, their equivalence is evident from the fact that their recursion trees are structurally identical - including the qualification of leaf nodes as 1-implicants, prime 1-implicants and prime 0-implicants. Also, the differences in the recursion trees of $\langle X_2 Y_2^2 Z_3^3 0^2 \rangle$ and $\langle X_2 Y_2^2 Z_3^3 1^2 \rangle$ in Fig. 1. make it clear that they do not implement the same Boolean function. This means that two Majority terms on the same set of variables are equivalent if and only if their recursion trees for Algorithm 1 are identical. Hence, the equivalence of two Majority terms can simply be determined by growing their recursion trees together as long as they form identical nodes, doing away with the need to generate the complete sum of all prime implicants.

Interestingly, this process can be made more efficient by observing the relationship between the recursion trees of dual Majority terms. Observe that, in Fig. 1, the recursion tree of the node $\langle X_2\, 1^7\, 0^8 \rangle$ is the mirror image of that of its dual $\langle X_2\, 1^8\, 0^7 \rangle$ along the horizontal axis with the red and green colors swapped. Similarly, the recursion tree of the node $\langle X_2\, Y_2^2\, 1^8\, 0^3 \rangle$ (in Fig. 1b) is the mirror image of that of its dual $\langle X_2\, Y_2^2\, 1^3\, 0^8 \rangle$ (in Fig. 1a) along the horizontal axis with the red and green colors swapped. The same is true for the node $\langle X_2\, Y_2^2\, 1^5\, 0^6 \rangle$ (in Fig. 1b) and its dual $\langle X_2\, Y_2^2\, 1^6\, 0^5 \rangle$ (in Fig. 1a). In fact, this is true for the whole

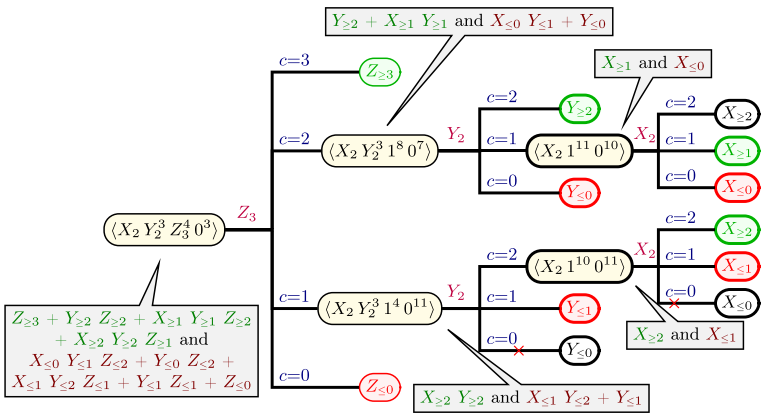


Fig. 2 The recursion tree generating all the prime implicants of the function h implemented by the Majority term $\langle X_2\, Y_2^3\, Z_3^4\, 0^3 \rangle$ is identical to that of $\langle X_2 Y_2^2 Z_3^3 0^2 \rangle$ in Fig. 1a

recursion trees in Fig. 1b and a, which represent the dual Majority terms $\langle X_2 Y_2^2 Z_3^3 0^2 \rangle$ and $\langle X_2 Y_2^2 Z_3^3 1^2 \rangle$.

This leads us to examine the relationship between the implicants of a Majority term and its dual.

5.1 Duality of Implicants

We have noticed that there exists symmetry between the set of all prime implicants of a Majority term and that of its dual. We see in Lemma 2 that there exists a one-to-one correspondence between the 1-implicants of a Majority term and the 0-implicants of its dual.

Lemma 2 (**Duality of Implicants**) *Consider a function F as per **Scenario A**. The product term ${}^{k_1-}W_{\geq c_{k_1}}\ {}^{k_2-}W_{\geq c_{k_2}} \cdots {}^{k_s-}W_{\geq c_{k_s}}$ is a 1-implicant of F if and only if the product term ${}^{k_1-}W_{\leq (n_1-c_{k_1})}\ {}^{k_2-}W_{\leq (n_2-c_{k_2})}\ {}^{k_s-}W_{\leq (n_s-c_{k_s})}$ is a 0-implicant of its dual $F^d = \langle {}^{1-}W_{n_1}^{l_1}\ {}^{2-}W_{n_2}^{l_2} \cdots {}^{m-}W_{n_m}^{l_m} 1^q 0^p \rangle$. Both of these implicant terms represent the same constraints on the variables.*

Proof Both the functions F and F^d are implemented by Majority-terms on multisets of the same cardinality, and hence need the same lim number of '1's (or '0's) to evaluate to '1' (or '0').

According to Theorem 1, the product term ${}^{k_1-}W_{\geq c_{k_1}}\ {}^{k_2-}W_{\geq c_{k_2}} \cdots {}^{k_s-}W_{\geq c_{k_s}}$ is a 1-implicant of F if and only if

$$c_{k_1} l_{k_1} + c_{k_2} l_{k_2} + \cdots + c_{k_s} l_{k_s} + p \geq lim$$

Similarly, according to Theorem 2, ${}^{k_1-}W_{\leq (n_1-c_{k_1})}\ {}^{k_2-}W_{\leq (n_2-c_{k_2})}\ {}^{k_s-}W_{\leq (n_s-c_{k_s})}$ is a 0-implicant of F^d if and only if

$$(n_{k_1} - (n_{k_1} - c_{k_1}))l_{k_1} + (n_{k_2} - (n_{k_2} - c_{k_2}))l_{k_2} + \cdots + (n_{k_s} - (n_{k_s} - c_{k_s}))l_{k_s} + p \geq lim$$

These two conditions are identical. Thus proved. □

Naturally, as noted in Theroem 6, such a one-to-one correspondence exists between the prime 1-implicants of a Majority term and the prime 0-implicants of its dual too.

Theorem 6 (**Duality of Prime Implicants**) *Consider a function F as per **Scenario A**. The product ${}^{k_1-}W_{\geq c_{k_1}}\ {}^{k_2-}W_{\geq c_{k_2}} \cdots {}^{k_s-}W_{\geq c_{k_s}}$ is a prime 1-implicant of F if and only if the product term ${}^{k_1-}W_{\leq (n_1-c_{k_1})}\ {}^{k_2-}W_{\leq (n_2-c_{k_2})}\ {}^{k_s-}W_{\leq (n_s-c_{k_s})}$ is a prime 0-implicant of its dual $F^d = \langle {}^{1-}W_{n_1}^{l_1}\ {}^{2-}W_{n_2}^{l_2} \cdots {}^{m-}W_{n_m}^{l_m} 1^q 0^p \rangle$.*

Proof Without loss of generality, it can be assumed that l_{k_1} is the least of the multiplicities l_{k_i}. According to Theorem 3, the product $T = {}^{k_1-}W_{\geq c_{k_1}}\ {}^{k_2-}W_{\geq c_{k_2}} \cdots {}^{k_s-}W_{\geq c_{k_s}}$ is a prime 1-implicant of the Boolean function F if and only if it is a 1-implicant of F but the product term $T' = {}^{k_1-}W_{\geq (c_{k_1}-1)}\ {}^{k_2-}W_{\geq c_{k_2}}\ {}^{k_3-}W_{\geq c_{k_3}} \cdots {}^{k_s-}W_{\geq c_{k_s}}$ is not. As per Theorem 4, the product term $S = {}^{k_1-}W_{\leq (n_1-c_{k_1})}\ {}^{k_2-}W_{\leq (n_2-c_{k_2})}\ {}^{k_s-}W_{\leq (n_s-c_{k_s})}$ is a prime 0-implicant of F^d if and only if it is a 0-implicant of F^d but the term $S' = {}^{k_1-}W_{\leq (n_1-c_{k_1}+1)}\ {}^{k_2-}W_{\leq (n_2-c_{k_2})}\ {}^{k_3-}W_{\leq (n_3-c_{k_3})} \cdots {}^{k_s-}W_{\leq (n_s-c_{k_s})}$ is not. Additionally, according to Lemma 2, the term T is a 1-implicant of F if and only if the term S is a 0-implicant of F^d. Also, the term T' is not a 1-implicant of F if and only if the term S' is not a 0-implicant of F^d. Hence, the term T is a prime 1-implicant of F if and only if the term S is a prime 0-implicant of its dual F^d. □

The one-to-one correspondence between implicants of dual Majority terms and the structural symmetry that it causes between the associated recursion trees is interesting in itself. However, this duality of implicants has a greater significance. An implicant of a Majority term is a representation of a condition under which the term evaluates to '1' or '0'. We have described this condition in our discussion of implicants of a Majority term (Theroems 1 and 2). These conditions of 1-implicants and 0-implicants together represent the Boolean function implemented by the Majority term. The one-to-one correspondence between implicants of dual Majority terms suggests that these conditions derived from a Majority term are the same as those derived from its dual. In Theroem 7, we note that not only is this true, even the Majority term obtained by self-dualizing a Majority term (substituting the constant with a dummy variable) represents the same set of conditions.

Theorem 7 *Consider **Scenario A**. The set of all prime 1-implicants and prime 0-implicants of the function $F = \langle {}^{1-}W_{n_1}^{l_1}\ {}^{2-}W_{n_2}^{l_2} \cdots {}^{m-}W_{n_m}^{l_m} 0^r \rangle$ represents the same constraints on the variables in V as the set of all prime 1-implicants and prime 0-implicants of the function $G = \langle {}^{1-}W_{n_1}^{l_1}\ {}^{2-}W_{n_2}^{l_2} \cdots {}^{m-}W_{n_m}^{l_m} 1^r \rangle$. Also, for a Boolean variable x, the same constraints on the variables in V are represented by the set of all prime 1-implicants and prime 0-implicants of the function H formed by self-dualizing F with x:*

$$H = \langle {}^{1-}W_{n_1}^{l_1}\ {}^{2-}W_{n_2}^{l_2} \cdots {}^{m-}W_{n_m}^{l_m} x^r \rangle.$$

Proof According to Theorem 6, each term that is prime 1-implicant of F has a corresponding term that is a prime

0-implicant of G, and these terms pose the same constraints on the variables in V. This means that the set of all prime 1-implicants of F represent the same constraints on the variables in V as does the set of all prime 0-implicants of G. Similarly, each term that is prime 0-implicant of F has a corresponding term that is a prime 1-implicant of G, and these terms pose the same constraints on the variables in V. This means that the set of all prime 0-implicants of F represent the same constraints on the variables in V as does the set of all prime 1-implicants of G. Thus, the set of all prime 1-implicants and prime 0-implicants of F and G represent the same constraints on the variables in V.

The prime implicants of the function H can be obtained by using Theorem 5 with respect to the variable x. That is, the set of all prime implicants of F and G combined with the state of the variable x represent the function H. Disregarding the variable x, this means that set of all prime 1-implicants and prime 0-implicants of the function H represent the same constraints on the variables in V as those of functions F and G. □

5.2 Detecting Equivalence of Majority terms

We use this insight to construct an efficient recursive method that checks if two Majority terms on the same set of variables implement the same Boolean function. This function shown in Algorithm 2 works from the bottom up, similar to Algorithm 1. It begins with a single maxcount-term — preferably one on the variable group with the largest multiplicity in one of the Majority terms. It checks if it is an implicant of one or both the terms. If it is not, the algorithm includes more maxcount-terms to this product term recursively, checking at each point if the product is an implicant of one or both the terms. If the product term is a 1-implicant (or 0-implicant) of both the Majority terms at the level, or is a 1-implicant (or 0-implicant) of neither, then the two Majority terms might be equivalent. Otherwise, they implement different Boolean functions. The function checks for equivalence recursively until either a mismatch is encountered or all cases are exhausted.

Note that if the product ${}^{k_1-}W_{\leq c_{k_1}}\ {}^{k_2-}W_{\leq c_{k_2}}\cdots {}^{k_s-}W_{\leq c_{k_s}}$ is a 0-implicant of both the functions, then so is the product ${}^{k_1-}W_{\leq (c_{k_1}-1)}\ {}^{k_2-}W_{\leq c_{k_2}}\ {}^{k_3-}W_{\leq c_{k_3}}\cdots {}^{k_s-}W_{\leq c_{k_s}}$, and it need not be checked for being one. Also, this algorithm uses Theorem 7 to reduce the cases checked. If the Majority term at any level of recursion is clearly self-dual (has no constant elements), then this algorithm checks equivalence for only half the counts of '1's possible in the group of variables under consideration. The other half counts lead to duals of these cases, and can be dropped since they represent the same set of constraints on the variables as the first half.

Algorithm 2 Equivalence of two Majority terms on the same variables.

Require: The term does not have an absolute Majority of '1' or '0'
function EQ($\langle {}^{1-}W_{n_1}^{l_1}\ {}^{2-}W_{n_2}^{l_2}\cdots {}^{m-}W_{n_m}^{l_m}1^p0^q\rangle$, $\langle {}^{1-}W_{n_1}^{l'_1}\ {}^{2-}W_{n_2}^{l'_2}\cdots {}^{m-}W_{n_m}^{l'_m}1^p0^q\rangle$)
  $lim \leftarrow \frac{f+n_1l_1+n_2l_2+\cdots+n_ml_m+p+q+1}{2}$
  $lim' \leftarrow \frac{g+n_1l'_1+n_2l'_2+\cdots+n_ml'_m+p'+q'+1}{2}$
  $low \leftarrow 0$
  if $p = q$ and $p' = q'$ **then**
    $low \leftarrow ceil(\frac{n_m}{2})$
  end if
  for $c \leftarrow n_m$ to low **do**
    if $cl_m + p \geq lim$ and $cl'_m + p' \geq lim'$ **then**
      continue
    else if $cl_m + p \geq lim$ or $cl'_m + p' \geq lim'$ **then**
      return FALSE
    end if
    if $(n_m - c)l_m + q \geq lim$ and $(n_m - c)l'_m + q' \geq lim'$ **then**
      return TRUE
    else if $(n_m - c)l_m + q \geq lim$ or $(n_m - c)l'_m + q' \geq lim'$ **then**
      return FALSE
    end if
    $S \leftarrow \langle {}^{1-}W_{n_1}^{l_1}\cdots {}^{m-1}W_{n_{m-1}}^{l_{m-1}}1^{p+cl_m}0^q + (n_m - c)l_m\rangle$
    $T \leftarrow \langle {}^{1-}W_{n_1}^{l'_1}\cdots {}^{m-1}W_{n_{m-1}}^{l'_{m-1}}1^{p'+cl'_m}0^{q'} + (n_m - c)l'_m\rangle$
    if *not* EQ(S, T) **then**
      return FALSE
    end if
  end for
  return TRUE
end function

Algorithm 2 needs the two Majority terms that need to be checked to be in the form $\langle {}^{1-}W_{n_1}^{l_1}\ {}^{2-}W_{n_2}^{l_2}\cdots {}^{m-}W_{n_m}^{l_m}1^p0^q\rangle$ and $\langle {}^{1-}W_{n_1}^{l'_1}\ {}^{2-}W_{n_2}^{l'_2}\cdots {}^{m-}W_{n_m}^{l'_m}1^{p'}0^{q'}\rangle$. Any variable that occurs in one Majority term and not the other can be included with zero multiplicity in the other term. We provide for the inclusion of non-overlapping variables because there exist cases where some variables in a Majority term do not appear in its canonical form. For instance, $\langle a^2\, b^2\, c^2\, d\rangle$ is equivalent to $\langle a\, b\, c\rangle$.

In Fig. 3, we illustrate the workings of this function in verifying the equivalence of two pairs of Majority terms. These terms contain three groups of symmetric variables X_2, Y_2 and Z_1 with multiplicities one, two and three respectively. Hence, the recursion trees checking the equivalence of these terms grow to at most three levels — one level each for the three groups (X, Y, Z) in descending order of their multiplicities.

At each level, the algorithm checks if the terms match in behavior. It checks if any maxcount-term on the group of variables with the largest multiplicity is a 1-implicant of both or one of the terms being processed. For example, the green nodes in the second levels of both the recursion trees in Fig. 3 show that $Y_{\geq 2}$ and $Y_{\geq 1}$ are 1-implicants of both the Majority terms being processed. Next, it checks if any

mincount-term on the group of variables with the largest multiplicity is a 0-implicant of both or one of the terms being processed. For example, the green node for $c = 1$ in the third level of Fig. 3a shows that $X_{\leq 1}$ is a 0-implicant of both the Majority terms being processed. On the other hand, the red node in Fig. 3b shows that $Y_{\geq 0}$ is a 0-implicant of one of the Majority terms but not the other. For a given count k, the function makes a recursive call to check the equivalence of the Majority terms only when neither case is true. The function returns TRUE if it encounters no mismatch in the behaviors of the terms at any level, and gets TRUE from every recursive call it makes.

Thus, we have a method to check if two Majority terms on the same set of variables implement the same Boolean function that exploits the fundamental mathematics of Majority logic.

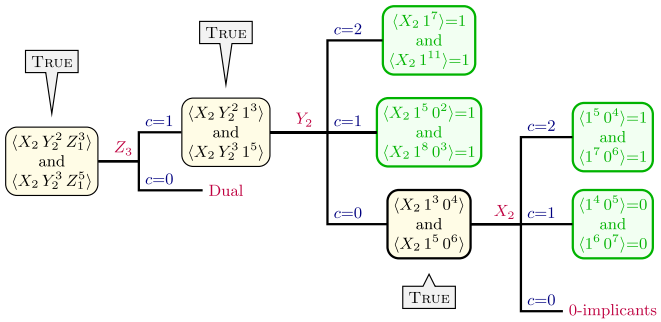
6 Compaction of n-Majority Term

We know that multiple Majority terms can implement the same Boolean function. It is critical in logic synthesis for Majority gates that equivalent Majority terms be identified and that the smallest possible equivalent of a Majority term be used. As we gear Majority algebra towards Majority logic synthesis, we seek to identify the Majority term on the smallest possible multiset that is equivalent to a given Majority term. We call it a compact Majority term, and examine how it can be derived from a Majority term.

We could use Algorithm 2 to check the equivalence of the given Majority term with many possible Majority terms until we find the compact Majority term. But such a brute force method is highly inefficient, and may not guarantee the Compact Majority term. Hence, we develop a method to compact a Majority term based on the knowledge of prime implicants and duality of implicants of Majority that we have developed thus far.

We know that the sum of all prime 1-implicants and the sum of all prime 0-implicants of a Boolean function identify it uniquely. We know the inequalities on the multiplicities of a Majority term that represent the prime implicants of a Majority term too (Theroem 3). This means that the compact Majority term may be obtained by solving the inequalities representing the set of all prime 1-implicants and prime 0-implicants of a Majority term with an objective to minimize the size of the multiset.

We have constructed a method to compact a Majority term using this idea. The COMPACT MAJORITY function in Algorithm 3 treats the problem of compacting a self-dual Majority term as that of finding new multiplicities for each group of symmetric variables in it. This function



(a) The Majority terms $\langle X_2\, Y_2^2\, Z_1^3\rangle$ and $\langle X_2\, Y_2^3\, Z_1^5\rangle$ implement the same Boolean function

(b) The Majority terms $\langle X_2\, Y_2^2\, Z_1^3\rangle$ and $\langle X_2\, Y_2^4\, Z_1^5\rangle$ implement different Boolean functions

Fig. 3 The equivalence of two Majority terms can be verified using Algorithm 2

assigns each group of symmetric variables ${}^{i-}W_{n_i}$ an integer variable l'_i to represent its new multiplicity. First, it specifies the current multiplicities of the Symmetry-groups as the upper limits of these variables. Next, it specifies zero as the lower limit of all multiplicities except the largest since the function implemented by the Majority term might be independent of some variables. It sets the lower limit of the variable representing the largest multiplicity to one since the Majority term does not have an absolute Majority on any variable. Further, it includes constraints that maintain the order of the multiplicities. Most importantly, it uses the CONSTRAINTS function to include the constraints representing the prime 1-implicants and prime 0-implicants of the Majority term.

The CONSTRAINTS function in Algorithm 3 begins with a self-dual Majority term and recursively constructs the inequalities using the l'_i variables. It treats l'_i as variables and lim', 1-*contrib*, and 0-*contrib* as expressions involving these variables, and builds a set *constraints* using these. At any level of recursion, the parameters 1-*contrib* and 0-*contrib* are expressions involving variables l'_i that represent the combination of the multiplicities and '1' counts that gave rise to the number of constant '1's and '0's in the current Majority term.

The CONSTRAINTS function follows the same pattern of recursion as the PRIME IMPLICANTS function in Algorithm 1, except that it trims the recursion tree using Theroem 7 as does Algorithm 2. Whenever a node represents a self-dual Majority term, the function visits only the top half of the recursion tree that would have been obtained using Algorithm 1. When the function encounters a prime implicant, it includes the appropriate inequality in *constraints*.

Though the constraints representing all prime implicants completely describe the logical behavior of the Majority term, their solution may not be able to represent a valid Majority term. A valid Majority term has an underlying multiset of odd cardinality. Hence, it is crucial that this oddness constraint be included at the first invocation of CONSTRAINTS. Also, any self-dual node that appears in the recursion tree has an equal number of '1's and '0's. This condition has to be added to the description of the Majority term. The CONSTRAINTS function includes a self-duality condition for each self-dual node that it encounters in the form of an equality between the expressions $1-contrib$ and $0-contrib$. Further, the number of non-constant elements in a self-dual Majority term is odd since the total number of constant elements is even. The CONSTRAINTS function includes an oddness condition on the non-constant elements for each self-dual node.

Algorithm 3 Compact a Majority term to a small multiset using constraints.

Require: The term does not have an absolute Majority of any variable.
Require: Multiplicities in ascending order: $1_i \le l_{i+1}, l'_i \le l'_{i+1}$ for all i
Require: There are no constants in ${}^{1-}W^{l_1}_{n_1}\ {}^{2-}W^{l_2}_{n_2}, \cdots,\ {}^{m-}W^{l_m}_{n_m}$
function COMPACT MAJORITY($\langle {}^{1-}W^{l_1}_{n_1}\ {}^{2-}W^{l_2}_{n_2} \cdots {}^{m-}W^{l_m}_{n_m}\rangle$)
  $constr \leftarrow \{l'_i \ge 0\} \uplus \{l'_m \ge 1\}$ ▷ Add limits on multiplicities
  $constr \leftarrow constr \uplus \{l'_i \le l_i\} \uplus \{l'_i \le l'_{i+1}\}$ ▷ Add ordering constraints
  $constr \leftarrow constr \uplus$ CONSTRAINTS($\langle {}^{1-}W^{l_1}_{n_1}\ {}^{2-}W^{l_2}_{n_2} \cdots {}^{m-}W^{l_m}_{n_m}\rangle$, "", "")
  Solve for l'_i under *constr* to minimize $n_1 l'_1 + n_2 l'_2 + \cdots + n_m l'_m$
  $compactTerm \leftarrow \langle {}^{1-}W^{l'_1}_{n_1}\ {}^{2-}W^{l'_2}_{n_2} \cdots {}^{m-}W^{l'_m}_{n_m}\rangle$
  return *compactTerm*
end function
function CONSTRAINTS($\langle {}^{1-}W^{l_1}_{n_1}\ {}^{2-}W^{l_2}_{n_2} \cdots {}^{m-}W^{l_m}_{n_m}\ {}^{1}P^{0}_{q}\rangle$, 1-*contr*,0-*contr*)
  $constr \leftarrow \{\}$ ▷ Set of constraints representing given Majority-term
  $lim \leftarrow \frac{n_1 l_1 + n_2 l_2 + \cdots + n_m l_m + p + q + 1}{2}$ ▷ No. needed for absolute Majority
  $lim' \leftarrow \frac{n_1 l'_1 + n_2 l'_2 + \cdots + n_m l'_m + 1 - contr + 0 - contr + 1}{2}$ ▷ Limit: For constraints
  $low \leftarrow 0$; $c \leftarrow n_m$
  if $p = q$ **then** ▷ Cardinality needs to be odd at this point
    $constr \leftarrow constr \uplus \{n_1 l'_1 + n_2 l'_2 + \cdots + n_m l'_m + 1\text{-}contr + 0\text{-}contr$ is odd$\}$
    $low \leftarrow ceil(\frac{n_m}{2})$ ▷ Check only half the dual cases (Theorem 7)
  end if
  while $c \ge low$ and $cl_m + p \ge lim$ **do** ▷ Skip 1-implicant (Th.1)
    $c \leftarrow c - 1$
  end while

  if $c < n_m$ **then** ▷ ${}^{m-}W_{\ge c+1}$ is a prime 1-implicant (Theorem 3)
    $constr \leftarrow constr \uplus \{(c+1)l'_m + 1\text{-}contr \ge lim'\}$
  end if
  while $c \ge low$ and $(n_m - c)l_m + q < lim$ **do**
    ▷ Find prime implicants recursively (Theorem 5)
    $sub\text{-}1\text{-}contr \leftarrow cl'_m + 1\text{-}contr$ ▷ Updated $1-contr$: For constraints
    $sub\text{-}0\text{-}contr \leftarrow (n_m - c)l'_m + 0\text{-}contr$ ▷ Updated 0-*contr*
    $subTerm \leftarrow \langle {}^{1-}W^{l_1}_{n_1}\ {}^{2-}W^{l_2}_{n_2} \cdots {}^{m+1}W^{l_{m+1}}_{n_{m+1}}\ 1^{p+cl_m}\ 0^{q+(n_m-c)l_m}\rangle$
    $subConstrs \leftarrow$ CONSTRAINTS($subTerm, sub\text{-}1\text{-}contr, sub\text{-}0\text{-}contr$)
    $constr \leftarrow constr \uplus subConstrs$
    if $cl_m + p = (n_m - c)l_m + q$ **then** ▷ *subTerm* is self-dual
      $constr \leftarrow constr \uplus \{cl'_m + 1\text{-}contr = (n_m - c)l'_m + 0\text{-}contr\}$
    end if
    $c \leftarrow c - 1$
  end while
  if $c \ge 0$ **then** ▷ ${}^{m-1}W_{\le c}$ is a prime 0-implicant (Theorem 4)
    $constr \leftarrow constr \uplus \{(n_m - c)l'_m + 0\text{-}contr \ge lim'\}$
  end if
  return *constr*
end function

With these constraints, the COMPACT MAJORITY function solves for integer values of the new multiplicities l'_i subject to these constraints with an objective to minimize the cardinality of the multiset. This cardinality is dictated by the new multiplicities and the number of variables in each Symmetry-group.

In Fig. 4, we illustrate the workings of the COMPACT MAJORITY and CONSTRAINTS functions on the Majority term $\langle X_2^2\, Y_3^3\, Z_2^6\rangle$. The COMPACT MAJORITY begins by assigning the variables x, y and z to represent the new multiplicities of the sets X_2, Y_3 and Z_2 respectively. It

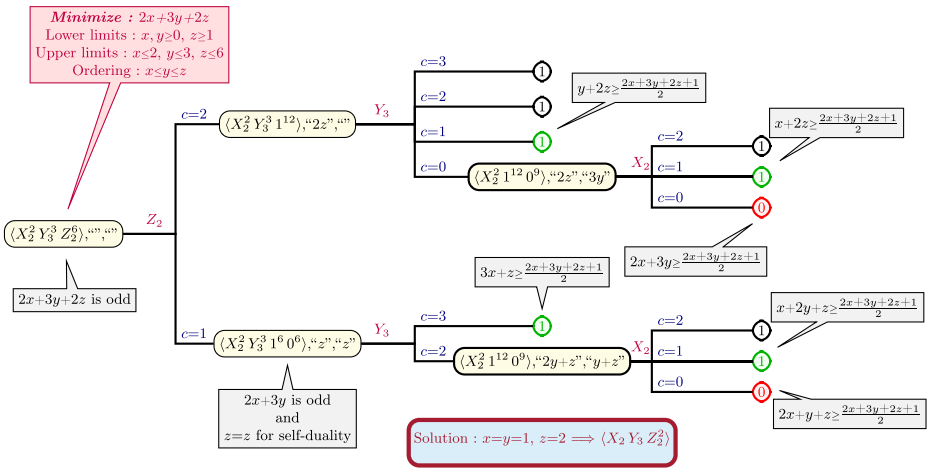


Fig. 4 The majority term $\langle X_2^2\, Y_3^3\, Z_2^6$ can be compacted to $\langle X_2\, Y_3\, Z_2^2\rangle$ using Algorithm 3

builds the constraints representing the bounds on these variables and their ordering. It then calls the CONSTRAINTS function with empty strings for the parameters $1{-}contrib$ and $0{-}contrib$. The CONSTRAINTS function checks for 1-implicants and 0-implicants at each level and adds the appropriate constraint when it encounters a prime implicant. For example, the green nodes in Fig. 4 represent prime 1-implicant, and the corresponding constraint is built using the count of '1's under consideration and the $1{-}contrib$ parameter of the call. Similarly, the red node in Fig. 4 represents a prime 0-implicant, and the corresponding constraint is built using the count of '0's under consideration and the $0{-}contrib$ parameter of the call. If the node is not an implicant, the CONSTRAINTS function makes a recursive call with new $1{-}contrib$ and $0{-}contrib$ parameters.

The node for $c = 1$ in the first level of Fig. 4 is a self-dual node, and hence the function adds one constraint for oddness, and one for self-duality. Similarly, being on a self-dual term, the first call to CONSTRAINTS adds an oddness constraint. The COMPACT MAJORITY function solves for the variables x, y and z under the nine constraints thus generated by CONSTRAINTS and the ones it generated. It solves for the values of these variables with the objective to minimize $2x + 3y + z$.

The optimal solution of this system of constraints is $x = y = 1$ and $z = 2$. Thus, the Majority term $\langle X_2^2\, Y_3^3\, Z_2^6\rangle$ can be compacted to $\langle X_2\, Y_3\, Z_2^2\rangle$.

We now have an efficient method to minimize a Majority term. Our method produces "a" compact Majority term of the given Majority term. Whether it produces "the" compact Majority term of the given Majority term depends on the validity of three assumptions that we have made in the construction of this algorithm :

Uniqueness Every Majority term has a unique compact form — an equivalent Majority term on a multiset of minimal size. Uniqueness ensures that the compact Majority term is a canonical form of the Majority term, and can be used to verify equivalence.

Symmetry and Multiplicity If the function implemented by a Majority term is symmetric in two variables, then these variables have the same multiplicity in the compact Majority term. This allows us to use a single multiplicity variable to represent a group of symmetric variables rather than using one per variable.

Ordered Multiplicities If the multiplicity of a variable x is greater than or equal to that of a variable y in the Majority term, then the multiplicity of x is greater than or equal to that of y in the compact Majority term too. This reduces the search space and aids in the convergence of the solution of the constraints.

We have observed that all these three conditions are true for many Majority terms. We have *compacted* all possible n-input Majority terms for n up to 25. As evident from Table 2, more than half the terms could be minimized. Even so, we are yet to come across counter-examples that invalidate our conjectures. Hence, though we are yet to prove these conjectures, we have used them to construct an efficient algorithm to derive the compact Majority term of a given Majority term.

Note that, even if these assumptions are proved invalid, the algorithm proposed is still useful in Majority logic synthesis since it reduces the size of Majority terms efficiently.

7 Efficiency of Algorithms

We have proposed the first known algorithms that use insights from the fundamental mathematics of n-input to generate minimal prime implicants of a Majority term, to check equivalence of two Majority terms, and to compact an n-input term to minimize the cardinality of its underlying multiset.

Table 2 A large subset of all possible n-input terms for any given n can be written as smaller Majority terms

n	5	7	9	11	13	15	17	19	21	23	25
All n-Maj	3	8	18	37	71	131	230	393	653	1060	1686
$(< n)$-Maj	1	4	10	22	42	79	137	231	376	600	926

The only point of comparison that exists for our algorithms is the work by Gowda et. al [4]. The exact point of comparison is the TG2MFF algorithm that they propose to generate a maximally factored form of a threshold gate, since we do not deal with logic networks as they proceed to do. The TG2MFF algorithm is based on generating a binary recursion tree of cofactors by each variable. This means that their recursion tree is of the order $O(2^n)$ (one level per variable), whereas, all our algorithms work with recursion trees of the order $O(l_1.l_2.\cdots.l_m)$ where $n = l_1 + l_2 + \cdots + l_m$ (one level per Symmetry-group). Also, the number of terms (and hence, literals) in the count-term based prime implicants we generate are much smaller than those in their Maximally factored form.

Table 3 PRIME IMPLICANTS scales better than TG2MFF as the number of calls in its recursion tree grows exponentially with the number of Symmetry-groups (#S) in a Majority Term rather than with the number of variables (#V) in it

#V	#S	Random Maj-term #1				Random Maj-term #2			
		TG2MFF		PRIME IMPL		TG2MFF		PRIME IMPL	
		Calls	Time	Calls	Time	Calls	Time	Calls	Time
11	1	923	1ms	1	0ms	923	1ms	1	0ms
15	1	1.2e4	11ms	1	0ms	1.2e4	10ms	1	0ms
21	1	7.0e5	0.5s	1	0ms	7.0e5	0.5s	1	0ms
21	5	3.7e5	0.3ms	203	0ms	9.6e5	0.7s	217	0ms
21	10	5.7e4	43ms	815	0ms	5.7e5	0.4ms	1027	1ms
21	15	8.9e4	70ms	7233	9ms	7.8e5	0.6ms	6188	7ms
21	20	8.8e4	75ms	3.2e4	43ms	7.6e5	0.5s	3.0e4	38ms
25	5	9.6e5	0.7s	64	0ms	4.2e6	2s	63	0ms
25	10	1.6e6	1s	2347	2ms	9.5e6	6s	3592	4ms
25	15	8.2e5	0.6s	1.9e4	23ms	1.8e6	1s	1.8e4	25ms
25	20	2.1e6	1s	1.9e5	0.2s	1.2e7	8s	1.9e5	0.2s
31	5	2.1e8	2m	484	0ms	6.2e8	7m	549	0ms
31	10	1.4e7	9s	3311	5ms	3.1e7	21s	3551	5ms
31	20	4.2e7	29s	9.5e5	1s	5.4e8	6m	1.2e6	1s
35	25			1.0e7	11s			1.4e7	12s
45	20			7.3e7	56s			7.4e7	1m
45	25			2.2e8	3m			2.0e8	3m
45	30			1.8e9	32m			1.3e9	19m
51	20			2.1e8	2m			2.2e8	2m
61	20			7.8e8	10m			7.7e8	10m
65	15			1.1e8	1m			1.5e8	1m
71	10			4.6e6	3s			4.2e6	2s
75	10			7.7e6	5s			8.2e6	5s
75	15			8.4e8	11m			8.9e8	10m
85	5			1.1e4	7ms			1.1e4	8ms
85	10			4.0e7	25s			4.4e7	26s
85	15			8.2e8	10m			1.2e9	14m
95	5			6.6e4	45ms			6.8e4	46ms
95	10			2.3e8	2m			2.3e8	2m
101	5			1.3e4	14ms			1.4e4	13ms
101	10			9.5e7	1m			7.9e7	52s
1001	5			4.4e8	6m			4.6e8	6m

We ported the TG2MFF algorithm to Majority terms and compared its performance against our Algorithm 1 PRIME IMPLICANTS. Table 3 shows the run-time statistics for TF2MFF and PRIME IMPLICANTS on a PC with an Intel i7-6700HQ CPU @2.60GHz×8 processor and 16GB RAM. We generated two random n-input terms for each pair of number of variables ($\#V$), and number of Symmetry-groups($\#S$) (all variables in a group have the same multiplicity) that they are grouped into.

First, we ran TG2MFF and PRIME IMPLICANTS on all the terms generated to populate Table 3 . This table shows the number of recursive calls made to the algorithms and the time taken to complete execution for each Majority term. The entries for cases that took more than an hour to complete execution have been left blank. The number of calls in the recursion trees provide an indication of both the number of terms produced and the complexity of the algorithms. Note that TG2MFF and reaches this mark early on. This is because it the number of terms in the factored form that it generates grow exponentially with the total number of variables $\#V$, irrespective of the symmetry present in the Majority term. For instance, the number of TG2MFF calls (and factor terms) exceed 700000 for $\#V = 21$ and $\#S = 1$ as seen in row 3. This is merely the term $\langle W_{21} \rangle$ which PRIME IMPLICANTS treats trivially with just one call that returns $W_{\geq 11}$. However, they perform comparably for $\#V = 21$ and $\#S = 21$. Higher the symmetry present in a Majority term, the greater the disparity between the number of calls to TG2MFF and PRIME IMPLICANTS despite the same number of variables. While both algorithms do grow exponentially, PRIME IMPLICANTS does scale better than TG2MFF—for instance, the number of calls to PRIME IMPLICANTS for $\#V = 1001$ and $\#S = 5$ are comparable to those to TG2MFF for $\#V = 31$ and $\#S = 5$.

The complexity of the number of calls seen in PRIME IMPLICANTS apply to EQ and COMPACT MAJORITY algorithms as well as they follow the same recursion tree. Generally, the recursion tree is heavily pruned in EQ using the duality and one-to-one correspondence of prime implicants. For instance, the run-time for EQ algorithm on all the pairs of Majority terms from Table 3 were under 10ms. However, as the Majority terms tend to be logically closer to each other, the performances of EQ and PRIME IMPLICANTS tend to be close as well.

The scalability of the COMPACT MAJORITY algorithm depends not only the exponential nature of the number of prime implicants (that PRIME IMPLICANTS generates), but also on the ILP solver. For this reason, its runtime tended to explode faster than that of PRIME IMPLICANTS. COMPACT MAJORITY needed less than a microsecond to minimize any Majority term with less than 31 variables, but took around 120ms to minimize a term with $\#V = 31$ and $\#S = 20$.

8 Conclusion

In this paper, we have solved three fundamental problems of n-input Logic: a) Generating a sum of count-based Prime Implicants representation of an n-input term, b) Efficient equivalence checking of two n-input terms, and c) Efficient compaction of an n-input term into one with fewer inputs. We begin by posing the idea of 0-implicants and prime 0-implicants as duals to the traditional (1-)implicants and prime (1-)implicants of Boolean functions and that the sum of all prime 0-implicants too is a canonical form. We utilize the properties of Majority to specially observe the implicants of a Majority term. With implicants of a Majority term as the basis, we have presented a recursive algorithm the generates the sum of all prime 1,0-implicants of a given Majority term. Stepping further ahead with the help of duality in Majority terms and among implicants, we have presented a recursive algorithm that checks for logical equivalance between two given Majority terms. Finally, we have presented a recursive algorithm that finds a compact equivalent for a given n-input term by employing Symmetry-groups and one-to-one correspondence of prime implicants among dual Majority terms.

Furthur work based on results presented in this paper may take multiple directions: a quantitative study of the efficiency of the algorithms, a theoretical study of the recursion trees built by algorithms based on the mathematical results presented, extension of algorithms to Majority networks, determination of the optimality and canonicity of compact Majority terms etc.

References

1. Akeela R, Wagh MD (2011) A five-input majority gate in quantum-dot cellular automata. In: NSTI Nanotech, vol 2, pp 978–981
2. Devadoss R, Paul K, Balakrishnan M (2015) Majsynth : an n-input majority algebra based logic synthesis tool for quantum-dot cellular automata. In: Proc. 24th international workshop on logic synthesis, 2015. IWLS'15
3. Devadoss R, Paul K, Balakrishnan M (2019) Majority logic: prime implicants and n-input majority term equivalence. In: Proc. 32nd International conference on VLSI design and 2019 18th international conference on embedded systems (VLSID). IEEE, pp 464–469
4. Gowda T, Vrudhula S, Konjevod G (2007) Combinational equivalence checking for threshold logic circuits. In: Proceedings of the 17th ACM great lakes symposium on VLSI. ACM, pp 102–107
5. Hulgaard H, Williams PF, Andersen HR (1999) Equivalence checking of combinational circuits using boolean expression diagrams. IEEE Trans Comput-Aided Design Integrated Circ Sys 18(7):903–917
6. Kang W, Zhang Y, Wang Z, Klein JO, Chappert C, Ravelosona D, Wang G, Zhang Y, Zhao W (2015) Spintronics: emerging ultra-low-power circuits and systems beyond mos technology. J Emerg Technol Comput Syst 12(2):16:1–16:42
7. Muroga S (1971) Threshold logic and its applications, Wiley-Interscience, New York

8. Sharad M, Augustine C, Panagopoulos G, Roy K (2012) Proposal for neuromorphic hardware using spin devices. CoRR, Cornell University. arXiv:1206.3227
9. Zhang R, Walus K, Wang W, Jullien GA (2004) A method of majority logic reduction for quantum cellular automata. IEEE Transactions on Nanotechnology 3(4):443–450

Rajeswari Devadoss is a researcher working on developing the mathematics of Majority logic and logic synthesis techniques for emerging technologies. She obtained her B.Tech in Computer Science and Engg. from the National Institute of Technology Tiruchirapalli in 2008, and Ph.D. from the CSE Dept. of the Indian Institute of Technology Delhi in 2016. She was a post doctoral researcher at the Nanyang Technological University, Singapore.

Kolin Paul is a Professor in the Department of Computer Science and Engg. at I.I.T. Delhi. He obtainted his B.Tech in Electronics & Comm. Engg. from REC Silchar, MS from Jadavpur University, Kolkata, and Ph.D. in Computer Science from BEC, Kolkata. He has been involved in teaching and research in the areas of Hardware and Embedded Systems Security, Reconfigurable Computing with Emerging Technologies for the past two decades. Over the years, he has spent time working at Tallinn University, IIT Bombay, KTH, Stockholm, and the University of Bristol. He has published more than 100 conference and journal papers.

M. Balakrishnan is a Professor in the Department of Computer Science & Engineering at I.I.T. Delhi. He obtained his B.E.(Hons.) in Electronics & Electrical Engg. from BITS Pilani and Ph.D. from EE Dept. IIT Delhi in 1985. For more than three decades, he is involved in teaching and research in the areas of digital systems design, electronic design automation and embedded systems. He has supervised 10 Ph.D. students, more than hundred B.Tech. and M.Tech students and published nearly 100 conference and journal papers. More recently the focus of his work has been in development of affordable assistive devices for visually impaired. He is involved in a number of projects that can enhance safe mobility as well as assist in education. SmartCane is the first product from the ASSISTECH group that has been launched as a product