2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)

Majority Logic: Prime Implicants and n-input Majority Term Equivalence

Rajeswari Devadoss
Dept of CSE, IIT Delhi
rajidrc@gmail.com

Kolin Paul
Dept of CSE, IIT Delhi
kolin@cse.iitd.ac.in

M Balakrishnan
Dept of CSE, IIT Delhi
mbala@cse.iitd.ac.in

***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. Logic synthesis that exploits these technologies demand an understanding of the mathematics of n-input Majority terms. One of the problems that turn up in such a study is the checking of equivalence of two n-input Majority terms on the same set of variables. We provide an algorithm based on prime implicants as a solution to this problem.**

In this quest, we extend the concept of implicants to two cases - 1-implicants and prime 1-implicants that imply 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, both being canonical representations of Boolean functions.

As Majority and Threshold functions have been shown to be logically equivalent, our algorithms are applicable to Threshold functions as well. Also, being based on implicants of Majority, our algorithms improve on the known algorithm for equivalence checking for threshold logic terms.

***Index Terms*—Majority, Threshold, Prime Implicants, Canonical form, Equivalence Checking**

I. 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 [1], [2].As it can be converted into a 2-input AND (or OR) gate by setting one of its inputs to constant '1' (or '0'), not only does it pack more computational power than 2-input AND/OR gates, it forms a functionally complete set with the NOT gate. It has been well established that the 3-input Majority operation can be used to implement Boolean functions in compact forms [3]. The same can be said of its generalization, the n-input Majority operation [4].

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.*

While it is trivial to check the equivalence of two AND (or OR) terms, 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 $\langle a^5\, b^4\, c^2\rangle$, $\langle a^3\, b^3\, c^3\rangle$, $\langle a^2\, b^2\, c^2\, d\rangle$, and $\langle a\, b\, c\rangle$ are logically equivalent and represent the function ab+bc+ca. 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. As expected from a barely studied subject as the mathematics of n-input Majority operations, little is known about matters such as equivalence checking. In this paper, we create an efficient algorithm that checks for the logical equivalence of two n-input Majority terms using implicants of Majority. 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 produces ***two canonical representations of a given n-input Majority term - sums of prime 1,0-implicants***. Finally, we apply duality to prune this process and use it to check the equivalence of two n-input Majority terms efficiently.

To our knowledge, this is the first work that develops the idea of 1-implicants and 0-implicants, and presents the sum of all 1-implicants and sum of all 0-implicants as canonical forms, especially for Majority terms. Also, this is the only work as yet that approaches equivalence checking of Majority terms using implicants of Majority.

II. Related Work

It is well established that Majority gates and Threshold gates are logically equivalent and interchangeable. To our knowledge, the only work that has contributions towards presenting the Boolean function implemented by a Threshold function is by Gowda et. al [5]. 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 [6]. Instead of using a Maximally Factored form that can produce very large expressions of 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 gate and run BED based equivalence checking on the results. Our 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 flattened Majority/Threshold terms.

III. 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** is 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 does not cover a more specific 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. 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 its implicants and prime-implicants.

Let us define a scenario A for use in our considerations of implicants in this work: F is a function $F{=}\langle {}^{1\text{-}}W_{n_1}^{l_1}\, {}^{2\text{-}}W_{n_2}^{l_2} \cdots {}^{m\text{-}}W_{n_m}^{l_m}\, 1^p\, 0^q \rangle$ on a collection of Boolean variables formed by the set $V{=}\left\{ {}^{1\text{-}}W_{n_1}, {}^{2\text{-}}W_{n_2}, \cdots, {}^{m\text{-}}W_{n_m} \right\}$ where each ${}^{i\text{-}}W_{n_i}$ is the set of variables $\left\{w_{i_1}, w_{i_2}, \ldots, w_{i_{n_i}}\right\}$; and a subset of these variable sets $\left\{ {}^{k_1\text{-}}W_{n_{k_1}}, {}^{k_2\text{-}}W_{n_{k_2}}, \cdots, {}^{k_s\text{-}}W_{n_{k_s}} \right\} \subseteq V$.

Theorem 1 (1-implicants of Majority): Consider a function F as per scenario A. The minproduct term ${}^{k_1\text{-}}W_{\geq c_{k_1}}\, {}^{k_2\text{-}}W_{\geq c_{k_2}} \cdots {}^{k_s\text{-}}W_{\geq c_{k_s}}$ is a 1-implicant of F if and only if F evaluates to '1' when exactly $c_{k_1}, c_{k_2}, \ldots, c_{k_s}$ variables of the sets ${}^{k_1\text{-}}W_{n_{k_1}}, {}^{k_2\text{-}}W_{n_{k_2}}, \cdots, {}^{k_s\text{-}}W_{n_{k_s}}$ respectively are '1's. That is, ${}^{k_1\text{-}}W_{\geq c_{k_1}}\, {}^{k_2\text{-}}W_{\geq c_{k_2}} \cdots {}^{k_s\text{-}}W_{\geq c_{k_s}}$ 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

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

Theorem 2 (0-implicants of Majority): Consider a function F as per scenario A. The maxproduct term ${}^{k_1\text{-}}W_{\leq c_{k_1}}\, {}^{k_2\text{-}}W_{\leq c_{k_2}} \cdots {}^{k_s\text{-}}W_{\leq c_{k_s}}$ is a 0-implicant of F if and only if F evaluates to '0' when exactly $c_{k_1}, c_{k_2}, \ldots, c_{k_s}$ variables of the sets ${}^{k_1\text{-}}W_{n_{k_1}}, {}^{k_2\text{-}}W_{n_{k_2}}, \cdots, {}^{k_s\text{-}}W_{n_{k_s}}$ respectively are '1's. That is, ${}^{k_1\text{-}}W_{\leq c_{k_1}}\, {}^{k_2\text{-}}W_{\leq c_{k_2}} \cdots {}^{k_s\text{-}}W_{\leq c_{k_s}}$ 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

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

Theorem 3 (Prime Implicants of Majority): Assume l_{k_1} is the smallest of the multiplicities l_{k_i}. The term ${}^{k_1\text{-}}W_{\geq c_{k_1}}\, {}^{k_2\text{-}}W_{\geq c_{k_2}} \cdots {}^{k_s\text{-}}W_{\geq c_{k_s}}$ is a prime 1-implicant of F if and only if it implies F but ${}^{k_1\text{-}}W_{\geq (c_{k_1}-1)}\, {}^{k_2\text{-}}W_{\geq c_{k_2}}\, {}^{k_3\text{-}}W_{\geq c_{k_3}} \cdots {}^{k_s\text{-}}W_{\geq c_{k_s}}$ does not. Similarly, the term ${}^{k_1\text{-}}W_{\leq c_{k_1}}\, {}^{k_2\text{-}}W_{\leq c_{k_2}} \cdots {}^{k_s\text{-}}W_{\leq c_{k_s}}$ is a prime 0-implicant of F if and only if it implies $\overline{F}$ but ${}^{k_1\text{-}}W_{\leq (c_{k_1}+1)}\, {}^{k_2\text{-}}W_{\leq c_{k_2}}\, {}^{k_3\text{-}}W_{\leq c_{k_3}} \cdots {}^{k_s\text{-}}W_{\leq c_{k_s}}$ does not.

IV. Sum of all Prime Implicants of a Majority term

Having found a way to identify 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 ${}^{i\text{-}}W_{\geq k}$ implies a Majority-term if and only if its cofactor with respect to ${}^{i\text{-}}W_{\geq k}$ implies the Majority-term where k and n_i-k elements of ${}^{i\text{-}}W_{n_i}$ are substituted with '1' and '0' respectively. This can be used as the basis of recursion in implicants of a Majority term.

Theorem 4 (Recursion in Implicants): Consider a function F as per scenario A. The minproduct ${}^{k_1}\text{-}W_{\geq c_{k_1}}\,{}^{k_2}\text{-}W_{\geq c_{k_2}} \cdots {}^{k_s}\text{-}W_{\geq c_{k_s}}$ is a 1-implicant of F if and only if the minproduct term ${}^{k_2}\text{-}W_{\geq c_{k_2}}\,{}^{k_3}\text{-}W_{\geq c_{k_3}} \cdots {}^{k_{(s)}}\text{-}W_{\geq c_{k_{(s)}}}$ is a 1-implicant of the function G $\langle {}^{1}\text{-}W^{l_1}_{n_1} \cdots {}^{(k_1+1)}\text{-}W^{l_{(k_1+1)}}_{n_{(k_1+1)}} \cdots {}^{m}\text{-}W^{l_m}_{n_m}\; 1^{p+c_{k_1} l_{k_1}}\; 0^{q+(n_{k_1}\text{-}c_{k_1}) l_{k_1}} \rangle$ Similarly, the product ${}^{k_1}\text{-}W_{\leq c_{k_1}}\,{}^{k_2}\text{-}W_{\leq c_{k_2}} \cdots {}^{k_s}\text{-}W_{\leq c_{k_s}}$ is a 0-implicant of F if and only if the maxproduct term ${}^{k_2}\text{-}W_{\leq c_{k_2}}\,{}^{k_3}\text{-}W_{\leq c_{k_3}} \cdots {}^{k_s}\text{-}W_{\leq c_{k_s}}$ is a 0-implicant of G.

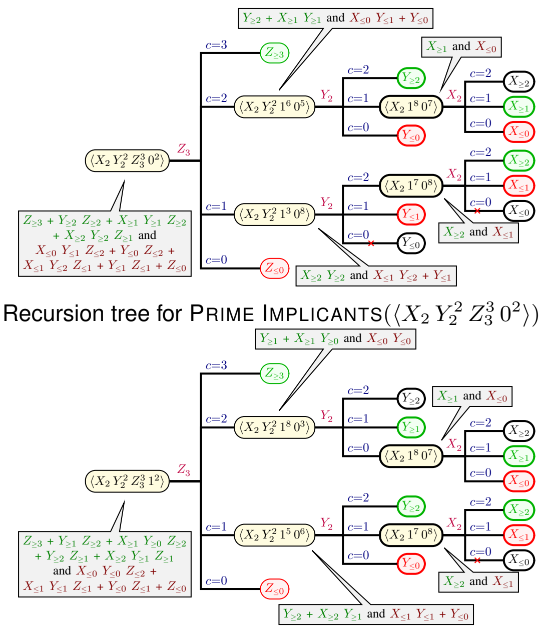
Consider that l_{k_1} is the smallest of the multiplicities l_{k_i}. The minproduct term ${}^{k_1}\text{-}W_{\geq c_{k_1}}\,{}^{k_2}\text{-}W_{\geq c_{k_2}} \cdots {}^{k_s}\text{-}W_{\geq c_{k_s}}$ is a prime 1-implicant of F if and only if the product term ${}^{k_2}\text{-}W_{\geq c_{k_2}}\,{}^{k_3}\text{-}W_{\geq c_{k_3}} \cdots {}^{k_s}\text{-}W_{\geq c_{k_s}}$ is a prime 1-implicant of G. Similarly, the maxproduct term ${}^{k_1}\text{-}W_{\leq c_{k_1}}\,{}^{k_2}\text{-}W_{\leq c_{k_2}} \cdots {}^{k_s}\text{-}W_{\leq c_{k_s}}$ is a prime 0-implicant of F if and only if the product term ${}^{k_1}\text{-}W_{\leq c_{k_1}}\,{}^{k_2}\text{-}W_{\leq c_{k_2}} \cdots {}^{k_{(s-1)}}\text{-}W_{\leq c_{k_{(s-1)}}}$ 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 $\langle {}^{1}\text{-}W^{l_1}_{n_1}\; {}^{2}\text{-}W^{l_2}_{n_2} \cdots {}^{m}\text{-}W^{l_m}_{n_m}\; 1^p\, 0^q \rangle$ 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 $W^{l_m}_{n_m}$. 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.

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

```
Require: Multiplicities in ascending order: $1_i \leq l_{i+1}$
Require: Term does not have an absolute Majority of ‘1’/‘0’
 1: function PIMPLS($\langle {}^{1}\text{-}W^{l_1}_{n_1}\; {}^{2}\text{-}W^{l_2}_{n_2} \cdots {}^{m}\text{-}W^{l_m}_{n_m}\; 1^p\, 0^q \rangle$)
 2:     $primeOne \leftarrow 0$                ▷ Sum of all prime 1-implicants
 3:     $primeZero \leftarrow 0$               ▷ Sum of all prime 0-implicants
 4:     $lim \leftarrow \frac{n_1 l_1 + n_2 l_2 + \cdots + n_m l_m + p + q + 1}{2}$
 5:     $c \leftarrow n_m$
 6:     while $c \geq 0$ and $c l_m + p \geq lim$ do
 7:         $c \leftarrow c\text{-}1$
 8:     end while
 9:     $primeOne \leftarrow primeOne + {}^{m}\text{-}W_{\geq c+1}$
10:     while $c \geq 0$ and $(n_m\text{-}c) l_m + q < lim$ do
11:         $subTerm \leftarrow \langle {}^{1}\text{-}W^{l_1}_{n_1}\; {}^{2}\text{-}W^{l_2}_{n_2} \cdots {}^{m\text{-}1}\text{-}W^{l_{m-1}}_{n_{m-1}}$
                $1^{p+c l_m}\, 0^{q+(n_m\text{-}c) l_m} \rangle$
12:         $subPrOne, subPrZero \leftarrow$ PIMPLS($subTerm$)
13:         $primeOne \leftarrow primeOne + subPrOne \cdot {}^{m}\text{-}W_{\geq c}$
14:         $primeZero \leftarrow primeZero + subPrZero \cdot {}^{m}\text{-}W_{\leq c}$
15:         $c \leftarrow c\text{-}1$
16:     end while
17:     $primeZero \leftarrow primeZero + {}^{m}\text{-}W_{\leq c}$
18:     return $primeOne, primeZero$
19: end function
```

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


$Y_{\geq 2}$ + $X_{\geq 1}$ $Y_{\geq 1}$ and $X_{\leq 0}$ $Y_{\leq 1}$ + $Y_{\leq 0}$
$X_{\geq 1}$ and $X_{\leq 0}$
$\langle X_2\, Y_2^2\, Z_3^3\, 0^2 \rangle$
Z_3
$\langle X_2\, Y_2^2\, 1^6\, 0^5 \rangle$
$\langle X_2\, Y_2^2\, 1^3\, 0^8 \rangle$
$\langle X_2\, 1^8\, 0^7 \rangle$
$\langle X_2\, 1^7\, 0^8 \rangle$
$Z_{\geq 3}$ + $Y_{\geq 2}$ $Z_{\geq 2}$ + $X_{\geq 1}$ $Y_{\geq 1}$ $Z_{\geq 2}$ + $X_{\geq 2}$ $Y_{\geq 2}$ $Z_{\geq 1}$ and $X_{\leq 0}$ $Y_{\leq 1}$ $Z_{\leq 2}$ + $Y_{\leq 0}$ $Z_{\leq 2}$ + $X_{\leq 1}$ $Y_{\leq 2}$ $Z_{\leq 1}$ + $Y_{\leq 1}$ $Z_{\leq 1}$ + $Z_{\leq 0}$
$X_{\geq 2}$ $Y_{\geq 2}$ and $X_{\leq 1}$ $Y_{\leq 2}$ + $Y_{\leq 1}$
$X_{\geq 2}$ and $X_{\leq 1}$
$Y_{\geq 1}$ + $X_{\geq 1}$ $Y_{\geq 0}$ and $X_{\leq 0}$ $Y_{\leq 0}$
$\langle X_2\, Y_2^2\, Z_3^3\, 1^2 \rangle$
$\langle X_2\, Y_2^2\, 1^8\, 0^3 \rangle$
$\langle X_2\, Y_2^2\, 1^5\, 0^6 \rangle$
$Z_{\geq 3}$ + $Y_{\geq 1}$ $Z_{\geq 2}$ + $X_{\geq 1}$ $Y_{\geq 0}$ $Z_{\geq 2}$ + $Y_{\geq 2}$ $Z_{\geq 1}$ + $X_{\geq 2}$ $Y_{\geq 1}$ $Z_{\geq 1}$ and $X_{\leq 0}$ $Y_{\leq 0}$ $Z_{\leq 2}$ + $X_{\leq 1}$ $Y_{\leq 1}$ $Z_{\leq 1}$ + $Y_{\leq 0}$ $Z_{\leq 1}$ + $Z_{\leq 0}$
$Y_{\geq 2}$ + $X_{\geq 2}$ $Y_{\geq 1}$ and $X_{\leq 1}$ $Y_{\leq 1}$ + $Y_{\leq 0}$

(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 Figure 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 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 Figure 1a, $Z_{\geq 3}$ is a prime 1-implicant while $Z_{\geq 2}$ is not. Next, it checks if any maxcount 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 Figure 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 Figure 1a, recursive calls are made for the cases that two and one elements of Z_3 are '1's.

When it makes a recursive call for count k on $i - W$, the function multiplies the sum of prime 1-implicants that it receives by ${}^{i\text{-}}W_{\geq k}$ 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 ${}^{i\text{-}}W_{\leq k}$ 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 Figure 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 $Z_{\leq 2}$ 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.

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.

V. 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 $\langle a\, b\, c\rangle$.

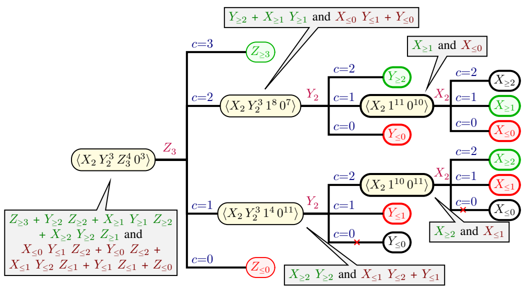

$Y_{\geq 2}$ + $X_{\geq 1}$ $Y_{\geq 1}$ and $X_{\leq 0}$ $Y_{\leq 1}$ + $Y_{\leq 0}$
$X_{\geq 1}$ and $X_{\leq 0}$
c=3
$Z_{\geq 3}$
c=2
$\langle X_2\, Y_2^3\, 1^8\, 0^7\rangle$
Y_2
c=2
$Y_{\geq 2}$
c=1
$\langle X_2\, 1^{11}\, 0^{10}\rangle$
X_2
c=2
$X_{\geq 2}$
c=1
$X_{\geq 1}$
c=0
$X_{\leq 0}$
c=0
$Y_{\leq 0}$
$\langle X_2\, Y_2^3\, Z_3^4\, 0^3\rangle$
Z_3
c=2
$\langle X_2\, 1^{10}\, 0^{11}\rangle$
X_2
c=2
$X_{\geq 2}$
c=1
$X_{\leq 1}$
c=0
$X_{\leq 0}$
c=1
$\langle X_2\, Y_2^3\, 1^4\, 0^{11}\rangle$
Y_2
c=1
$Y_{\leq 1}$
c=0
$Y_{\leq 0}$
$X_{\geq 2}$ and $X_{\leq 1}$
$Z_{\geq 3}$ + $Y_{\geq 2}$ $Z_{\geq 2}$ + $X_{\geq 1}$ $Y_{\geq 1}$ $Z_{\geq 2}$ + $X_{\geq 2}$ $Y_{\geq 2}$ $Z_{\geq 1}$ and $X_{\leq 0}$ $Y_{\leq 1}$ $Z_{\leq 2}$ + $Y_{\leq 0}$ $Z_{\leq 2}$ + $X_{\leq 1}$ $Y_{\leq 2}$ $Z_{\leq 1}$ + $Y_{\leq 1}$ $Z_{\leq 1}$ + $Z_{\leq 0}$
c=0
$Z_{\leq 0}$
$X_{\geq 2}$ $Y_{\geq 2}$ and $X_{\leq 1}$ $Y_{\leq 2}$ + $Y_{\leq 1}$

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 Figure 1a

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 - $\langle {}^{1\text{-}}W_{n_1}^{l_1}\; {}^{2\text{-}}W_{n_2}^{l_2}\; \cdots\; {}^{m\text{-}}W_{n_m}^{l_m}\; 1^p\, 0^q\rangle$ and $\langle {}^{1\text{-}}W_{n_1}^{l'_1}\; {}^{2\text{-}}W_{n_2}^{l'_2}\; \cdots\; {}^{m\text{-}}W_{n_m}^{l'_m}\; 1^{p'}\, 0^{q'}\rangle$. For example, as shown in Figure 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 Figure 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 $\langle X_2\, Y_2^2\, Z_3^3\, 1^2\rangle$ make it clear that they do not implement the same Boolean function Figure 1. 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 Figure 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 Figure 1b) is the mirror image of that of its dual $\langle X_2\, Y_2^2\, 1^3\, 0^8\rangle$ (in Figure 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 Figure 1b) and its dual $\langle X_2\, Y_2^2\, 1^6\, 0^5\rangle$ (in Figure 1a). In fact, this is true for the whole recursion trees in Figure 1b and Figure 1a, 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.

A. 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 Theorem 5 that there exists a one-to-one correspondence between the 1-implicants of a Majority-term and the 0-implicants of its dual.

Theorem 5 (Duality of Implicants): Consider a function F=$\langle {}^{1\text{-}}W_{n_1}^{l_1}\; {}^{2\text{-}}W_{n_2}^{l_2}\; \cdots\; {}^{m\text{-}}W_{n_m}^{l_m}\; 1^p\, 0^q\rangle$ as per scenario A. The product term ${}^{k_1\text{-}}W_{\geq c_{k_1}}\, {}^{k_2\text{-}}W_{\geq c_{k_2}} \cdots {}^{k_s\text{-}}W_{\geq c_{k_s}}$ is a 1-implicant of F if and only if the product term ${}^{k_1\text{-}}W_{\leq (n_1 - c_{k_1})}\, {}^{k_2\text{-}}W_{\leq (n_2 - c_{k_2})} \cdots {}^{k_s\text{-}}W_{\leq (n_s - c_{k_s})}$ is a 0-implicant of F^d=$\langle {}^{1\text{-}}W_{n_1}^{l_1}\; {}^{2\text{-}}W_{n_2}^{l_2}\; \cdots\; {}^{m\text{-}}W_{n_m}^{l_m}\; 1^q\, 0^p\rangle$. Both of these implicant terms represent the same constraints on the variables.

Naturally, as noted in Theorem 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 $F{=}\langle {}^{1\text{-}}W^{l_1}_{n_1}\, {}^{2\text{-}}W^{l_2}_{n_2} \cdots {}^{m\text{-}}W^{l_m}_{n_m}\, 1^p\, 0^q \rangle$ as per scenario A such that l_{k_1} is the smallest of the multiplicities l_{k_i}. The product term ${}^{k_1\text{-}}W_{\geq c_{k_1}}\, {}^{k_2\text{-}}W_{\geq c_{k_2}} \cdots {}^{k_s\text{-}}W_{\geq c_{k_s}}$ is a prime 1-implicant of F if and only if the product term ${}^{k_1\text{-}}W_{\leq (n_1\text{-}c_{k_1})}\, {}^{k_2\text{-}}W_{\leq (n_2\text{-}c_{k_2})} \cdots {}^{k_s\text{-}}W_{\leq (n_s\text{-}c_{k_s})}$ is a prime 0-implicant of $F^d{=}\langle {}^{1\text{-}}W^{l_1}_{n_1}\, {}^{2\text{-}}W^{l_2}_{n_2} \cdots {}^{m\text{-}}W^{l_m}_{n_m}\, 1^q\, 0^p \rangle$.

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 (Theorems 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 Theorem 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 a collection of Boolean variables formed by the set $V{=}\left\{{}^{1\text{-}}W_{n_1}, {}^{2\text{-}}W_{n_2}, \cdots, {}^{m\text{-}}W_{n_m}\right\}$ where each ${}^{i\text{-}}W_{n_i}$ is a set of Boolean variables $\{w_{i_1}, w_{i_2}, \ldots, w_{i_n}\}$. The set of all prime 1-implicants and prime 0-implicants of a function $F{=}\langle {}^{1\text{-}}W^{l_1}_{n_1}\, {}^{2\text{-}}W^{l_2}_{n_2} \cdots {}^{m\text{-}}W^{l_m}_{n_m}\, 0^r \rangle$ represents the same constraints on the variables in V as the set of all prime 1-implicants and prime 0-implicants of the function $G{=}\langle {}^{1\text{-}}W^{l_1}_{n_1}\, {}^{2\text{-}}W^{l_2}_{n_2} \cdots {}^{m\text{-}}W^{l_m}_{n_m}\, 1^r \rangle$. 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 $H{=}\langle {}^{1\text{-}}W^{l_1}_{n_1}\, {}^{2\text{-}}W^{l_2}_{n_2} \cdots {}^{m\text{-}}W^{l_m}_{n_m}\, x^r \rangle$.

B. 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 - one on the variable group with the largest multiplicity. It checks if it is an implicant of one or both the terms. If it is not, the algorithm includes more maxcounts 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.

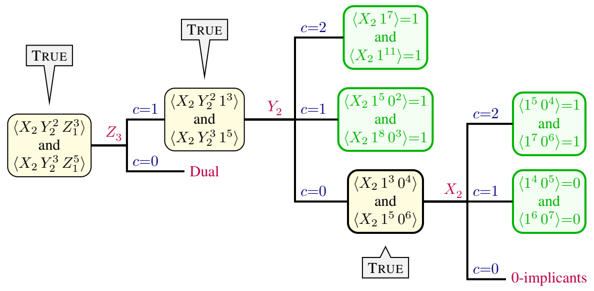
Algorithm 2 Equivalence of two Majority-terms on the same variables

```
Require: The term does not have an absolute Majority of '1'/'0'
function EQUIVALENT($\langle X_f\, {}^{1\text{-}}W^{l_1}_{n_1}\, {}^{2\text{-}}W^{l_2}_{n_2} \cdots {}^{m\text{-}}W^{l_m}_{n_m}\, 1^p\, 0^q \rangle$, $\langle Y_g\, {}^{1\text{-}}W^{l'_1}_{n_1}\, {}^{2\text{-}}W^{l'_2}_{n_2} \cdots {}^{m\text{-}}W^{l'_m}_{n_m}\, 1^{p'}\, 0^{q'} \rangle$)
    if got $\langle X_f\, 1^p\, 0^q \rangle$ and $\langle Y_g\, 1^{p'}\, 0^{q'} \rangle$ then
        return FALSE
    end if
    $lim \leftarrow \frac{f+n_1 l_1+n_2 l_2+\cdots+n_m l_m+p+q+1}{2}$
    $lim' \leftarrow \frac{g+n_1 l'_1+n_2 l'_2+\cdots+n_m l'_m+p'+q'+1}{2}$
    $low \leftarrow 0$
    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
            return TRUE
        else if $cl_m{+}p{\geq}lim$ or $cl'_m{+}p'{\geq}lim'$ then
            return FALSE
        end if
        if $(n_m\text{-}c)l_m{+}q{\geq}lim$ and $(n_m\text{-}c)l'_m{+}q'{\geq}lim'$ then
            return TRUE
        else if $(n_m\text{-}c)l_m{+}q{\geq}lim$ or $(n_m\text{-}c)l'_m{+}q'{\geq}lim'$ then
            return FALSE
        end if
        $S \leftarrow \langle X_f\, {}^{1\text{-}}W^{l_1}_{n_1} \cdots {}^{m\text{-}1\text{-}}W^{l_{m\text{-}1}}_{n_{m\text{-}1}}\, 1^{p+cl_m}\, 0^{q+(n_m\text{-}c)l_m} \rangle$
        $T \leftarrow \langle Y_g\, {}^{1\text{-}}W^{l'_1}_{n_1} \cdots {}^{m\text{-}1\text{-}}W^{l'_{m\text{-}1}}_{n_{m\text{-}1}}\, 1^{p'+cl'_m}\, 0^{q'+(n_m\text{-}c)l'_m} \rangle$
        if not EQUIVALENT$(S, T)$ then
            return FALSE
        end if
    end for
    return TRUE
end function
```

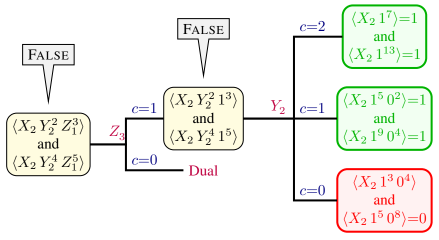
Our algorithm needs the Majority-terms to be in form $\langle X_f\, {}^{1\text{-}}W^{l_1}_{n_1}\, {}^{2\text{-}}W^{l_2}_{n_2} \cdots {}^{m\text{-}}W^{l_m}_{n_m}\, 1^p\, 0^q \rangle$ and $\langle Y_s\, {}^{1\text{-}}W^{l'_1}_{n_1}\, {}^{2\text{-}}W^{l'_2}_{n_2} \cdots {}^{m\text{-}}W^{l'_m}_{n_m}\, 1^{p'}\, 0^{q'} \rangle$, where X_f and Y_g are all the variables that do not overlap between the terms. 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 $\langle a\, b\, c \rangle$. We use a recursive approach based on implicants. If we see a function call with only non-overlapping variables, the terms are not equivalent.

Note that if the product ${}^{k_1\text{-}}W_{\leq c_{k_1}}\, {}^{k_2\text{-}}W_{\leq c_{k_2}} \cdots {}^{k_s\text{-}}W_{\leq c_{k_s}}$ is a 0-implicant of both the functions, then so is the product ${}^{k_1\text{-}}W_{\geq (c_{k_1}\text{-}1)}\, {}^{k_2\text{-}}W_{\geq c_{k_2}}\, {}^{k_3\text{-}}W_{\geq c_{k_3}} \cdots {}^{k_s\text{-}}W_{\geq c_{k_s}}$, and it need not be checked for being one. Also, it 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.

In Figure 3, we illustrate the workings of this function in verifying the equivalence of two pairs of Majority-terms.


TRUE
$\langle X_2\,Y_2^2\,Z_1^3\rangle$ and $\langle X_2\,Y_2^3\,Z_1^5\rangle$
Z_3
c=1
c=0
Dual
TRUE
$\langle X_2\,Y_2^2\,1^3\rangle$ and $\langle X_2\,Y_2^3\,1^5\rangle$
Y_2
c=2
$\langle X_2\,1^7\rangle$=1 and $\langle X_2\,1^{11}\rangle$=1
c=1
$\langle X_2\,1^5\,0^2\rangle$=1 and $\langle X_2\,1^8\,0^3\rangle$=1
c=0
$\langle X_2\,1^3\,0^4\rangle$ and $\langle X_2\,1^5\,0^6\rangle$
TRUE
X_2
c=2
$\langle 1^5\,0^4\rangle$=1 and $\langle 1^7\,0^6\rangle$=1
c=1
$\langle 1^4\,0^5\rangle$=0 and $\langle 1^6\,0^7\rangle$=0
c=0
0-implicants

(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


FALSE
$\langle X_2\,Y_2^2\,Z_1^3\rangle$ and $\langle X_2\,Y_2^4\,Z_1^5\rangle$
Z_3
c=1
c=0
Dual
FALSE
$\langle X_2\,Y_2^2\,1^3\rangle$ and $\langle X_2\,Y_2^4\,1^5\rangle$
Y_2
c=2
$\langle X_2\,1^7\rangle$=1 and $\langle X_2\,1^{13}\rangle$=1
c=1
$\langle X_2\,1^5\,0^2\rangle$=1 and $\langle X_2\,1^9\,0^4\rangle$=1
c=0
$\langle X_2\,1^3\,0^4\rangle$ and $\langle X_2\,1^5\,0^8\rangle$=0

(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

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 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 Figure 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 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 Figure 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 Figure 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 an 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.

VI. Efficiency of Algorithms

We have proposed the first known algorithms that use insights from the fundamental mathematics of n-input Majority to generate compact prime implicants of a Majority term and to check equivalence of two Majority terms. The only point of comparison that exists for our algorithms is the work by Gowda et. al [5]. 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 $O(2^n)$, whereas, both our algorithms work with recursion trees of the order $O(l_1{\cdot}l_2{\cdot}\cdots{\cdot}l_m)$ where $n{=}l_1{+}l_2{+}\cdots{+}l_m$. Also, the number of terms (and hence, literals) in the prime implicants we generate are much smaller than those in their Maximally factored form.

VII. Conclusion

In this paper, we have solved the fundamental problem of equivalence checking of generic n-input Majority terms efficiently. 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. The proofs of the theorems stated have been left out for lack of space in this paper.

References

[1] W. Kang, Y. Zhang, Z. Wang, J.-O. Klein, C. Chappert, D. Ravelosona, G. Wang, Y. Zhang, and W. Zhao, "Spintronics: Emerging ultra-low-power circuits and systems beyond mos technology," *J. Emerg. Technol. Comput. Syst.*, vol. 12, no. 2, pp. 16:1–16:42, Sep 2015.

[2] M. Sharad, C. Augustine, G. Panagopoulos, and K. Roy, "Proposal for neuromorphic hardware using spin devices," *CoRR*.

[3] R. Zhang, K. Walus, W. Wang, and G. A. Jullien, "A method of majority logic reduction for quantum cellular automata," *Nanotechnology, IEEE Transactions on*, vol. 3, no. 4, pp. 443–450, 2004.

[4] R. Devadoss, K. Paul, and M. Balakrishnan, "Majsynth : An n-input majority algebra based logic synthesis tool for quantum-dot cellular automata," in *Logic Synthesis, 2015. IWLS'15. 24th International Workshop on*, 2015.

[5] T. Gowda, S. Vrudhula, and G. Konjevod, "Combinational equivalence checking for threshold logic circuits," in *Proceedings of the 17th ACM Great Lakes symposium on VLSI*. ACM, 2007, pp. 102–107.

[6] H. Hulgaard, P. F. Williams, and H. R. Andersen, "Equivalence checking of combinational circuits using boolean expression diagrams," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 7, pp. 903–917, 1999.