Majority Logic Representation and Satisfiability

Luca Amarú, Pierre-Emmanuel Gaillardon, Giovanni De Micheli
Integrated Systems Laboratory (LSI), EPFL, Switzerland

***Abstract*—Majority logic is a powerful generalization of common AND/OR logic. Original two-level and multi-level logic networks can use majority operators as primitive connective, in place of AND/ORs. In such a way, Boolean functions have novel means for compact representation and efficient manipulation. In this paper, we focus on two-level logic representation. We define a *Majority Normal Form* (MNF), as an alternative to traditional *Disjunctive Normal Form* (DNF) and *Conjunctive Normal Form* (CNF). After a brief investigation on the MNF expressive power, we study the problem of MNF-*SATisfiability* (MNF-SAT). We prove that MNF-SAT is NP-complete, as its CNF-SAT counterpart. However, we show practical restrictions on MNF formula whose satisfiability can be decided in polynomial time. We finally propose a simple algorithm to solve MNF-SAT, based on the intrinsic functionality of two-level majority logic. Although an automated MNF-SAT solver is still under construction, manual examples already demonstrate promising opportunities.**

I. Introduction

Boolean logic is usually defined in terms of primitive AND ($\wedge$), OR ($\vee$) and INV ($'$) operators. Such formulation acts in accordance with the natural way logic designers interpret Boolean functions. For this reason, it emerged as a standard in the field. However, no evidence is provided that this formulation, or another, has the most efficient set of primitives for Boolean logic. In computer science, the efficiency of Boolean logic applications is measured by different metrics such as (i) the result quality, for example the performance of an automatically synthesized digital circuit, (ii) the runtime and (iii) the memory footprint of a software tool. With the aim to optimize them, the accordance to a specific logic model is no longer important. Indeed, the key is the expressive power of the set of primitives provided to the computer, that determines the capability to reach better metrics. Recently, majority logic has shown the opportunity to enhance the efficiency of multi-level logic optimization [1], [2] and reversible quantum logic synthesis [3].

In this paper, we extend the intuition provided in [2] to two-level logic and Boolean satisfiability. We provide an alternative two-level representation of Boolean functions based entirely on majority and complementation operators. We call it *Majority Normal Form* (MNF), using a similar notation as for traditional *Disjunctive Normal Form* (DNF) and *Conjunctive Normal Form* (CNF) [4]. The MNF can represent any Boolean function, therefore being *universal*, as CNF and DNF. We investigate then the satisfiability of MNF formula (MNF-SAT). In its most general definition, MNF-SAT is NP-complete, as its CNF-SAT counterpart. However, there exist interesting restrictions of MNF whose satisfiability can instead be decided in polynomial time. We finally propose an algorithm to solve MNF-SAT exploiting the nature of two-level majority logic. Even though an automated solver is still under construction, manual examples on such algorithm already demonstrate promising opportunities.

The study of majority logic is also motivated by the advance of emerging technologies. In the quest for increasing computational performance per area unit [15], majority/minority logic gates are natively implemented in different nanotechnologies [16]–[18]. In this scenario, a logic model based on majority primitives unlocks the full potential led by nanodevices. In this paper, we focus on abstract logic applications, to first showcase the intrinsic power of majority logic.

The remainder of this paper is organized as follows. Section II provides relevant background and notations. In Section III, the two-level *Majority Normal Form* is introduced and its features investigated. Section IV studies the satisfiability of MNF formula, from a theoretical perspective. Section V proposes a simple algorithm to solve MNF-SAT exploiting the intrinsic functionality of two-level majority logic. Section VI discusses future research directions. Section VII concludes the paper.

II. Background and Motivation

This section presents a brief background on two-level logic representation and Boolean satisfiability. Notations and definitions used in the rest of this paper are also introduced.

A. Notations and Definitions

In the Boolean domain, all variables belong to $\mathbb{B} = \{0, 1\}$. The *on*-set of a Boolean function is the set of input patterns evaluating the function to logic 1. Similarly, the *off*-set of a Boolean function is the set of input patterns evaluating the function to logic 0. Literals are variables and complemented ($'$) variables. Terms are conjunctions ($\wedge$) of literals. Clauses are disjunctions ($\vee$) of literals. A majority function of n (odd) literals returns the Boolean value most frequently appearent among the inputs. In the context of this paper, we refer to a *threshold* function as to a majority function with repeated literals. Note that this is a restriction of the more general definition of *threshold* functions [5].

B. Two-level Logic Representation

Traditional two-level logic representation combines terms and clauses to describe Boolean functions. A *Conjunctive Normal Form* (CNF) is a conjunction of clauses. A *Disjunctive Normal Form* (DNF) is disjunctions of terms. Both CNF and DNF are *universal* logic representation form, i.e., any Boolean function can be represented by them. For more information about logic representation forms, we refer the reader to [5].

C. Satisfiability

The Boolean *SATisfiability* problem (SAT) consists of determining whether there exists or not an assignment of variables so that a Boolean formula evaluates to true. SAT is a difficult problem for CNF formula. Indeed, CNF-SAT was the first known NP-complete problem [6]. Instead, DNF-SAT is trivial to solve [7]. Unfortunately, converting a CNF into a DNF, or viceversa, may require an exponential number of operations. Some restrictions of CNF-SAT, e.g., 2-SAT, Horn-SAT, XOR-SAT, etc., can be solved in polynomial time. For more information about SAT, we refer the reader to [7].

III. Two-Level Majority Representation Form

In this section, we present a two-level majority logic representation form as extension to traditional two-level conjunctive and disjunctive normal forms.

A. Majority Normal Form Definition and Properties

Both CNF and DNF formula require at least two Boolean operators, $\wedge$ and $\vee$, apart from the complementation. Interestingly enough, the majority includes both $\wedge$ and $\vee$ into a unique operator. This feature is formalized in the following.

Property The n-input (odd) majority operator filled with $\lfloor n/2 \rfloor$ logic zeros collapses into an $\lceil n/2 \rceil$-input $\wedge$ operator. Conversely, if filled with $\lfloor n/2 \rfloor$ logic ones it collapse into an $\lceil n/2 \rceil$-input $\vee$ operator.

Example Consider the function $M(a,b,c,0,0)$. Owing to the majority functionality, to evaluate such function to logic 1 all variables (a,b,c) must be logic 1. This is because already 2 inputs over 5 are fixed to logic 0, which is close to the majority threshold. Indeed, if even only one variable among (a,b,c) is logic 0, the function evaluates to 0. This is equivalent to the function $a \wedge b \wedge c$. Using a similar reasoning, $M(a,b,c,1,1)$ is functionally equivalent to $a \vee b \vee c$.

This remarkable property motivates us to define a novel two-level logic representation form.

Definition A *Majority Normal Form* (MNF) is a majority of majorities, where majorities are fed with literals, 0 or 1.

Example An MNF is $M(M(a,b,1),M(a,b,c,0,e'),d')$. Another MNF, for a different Boolean function, is $M(a,0,c,M(a,b',c'),(a',1,c))$. The expression $M(M(M(a,b,c),d,e),e,f,g,h)$ is not an MNF as it contains three levels of majority operators, while MNF is a two-level representation form.

Following its definition, MNF includes also CNF and DNF.

Property Any CNF (DNF) is structurally equivalent to an MNF, where the n-input conjunction (disjunction) is a majority operator filled by $\lfloor n/2 \rfloor$ logic zeros (ones) and by n clauses (terms) of m-inputs, that are themselves majority operators filled by $\lfloor m/2 \rfloor$ logic ones (zeros) and m literals.

We give hereafter an example of CNF to MNF translation.

Example The starting CNF is $(c' \vee b) \wedge (a' \vee c) \wedge (a \vee b)$. The $\wedge$ in the CNF is translated as $M(-,-,-,0,0)$. The clauses are instead translated in the form $M(-,-,1)$. The resulting MNF is $M(M(c',b,1),M(a',c,1),M(a,b,1),0,0)$.

It is straightforward now to show that CNF and DNF can be translated into MNF in linear time. However, the inverse translation of MNF into CNF or DNF can be more complex, as MNF are intrisically more expressive than CNF and DNF.

The MNF is a *universal* logic representation form, i.e., any Boolean function can be represented with it. This comes as a consequence of the inclusion of *universal* CNF and DNF. In addition to the emulation of traditional conjunction and disjunction operators, a majority operator features other noteworthy properties. First, majority is a *self-dual* function [5], i.e., the complement of a majority equals to the majority with complemented inputs. The *self-dual* property also holds when variables are repeated inside the majority operator (*threshold function*). Second, the majority is *fully-symmetric*, i.e., any permutation of inputs does not change the function behavior. In addition, the n-input majority where two inputs are one the complement of the other, collapses into a $(n\text{-}2)$-input majority. In order to extend the validity of these properties, it is proper to define $M(a) = a$, which is a majority operator of a single input, equivalent to a logic buffer.

B. Representation Examples with DNF, CNF and MNF

We provide hereafter some examples of MNF in contrast to their corresponding CNF and DNF.

Example Boolean function $a \vee (b \wedge c)$. The form $a \vee (b \wedge c)$ is already a DNF. A CNF is $(a \vee b) \wedge (a \vee c)$. An MNF is $M(a,1,M(0,b,c))$. Another, more compact, MNF is $M(a,b,c,a,1)$.

For the sake of illustration, Fig. 1 depicts the previous example by means of drawings.

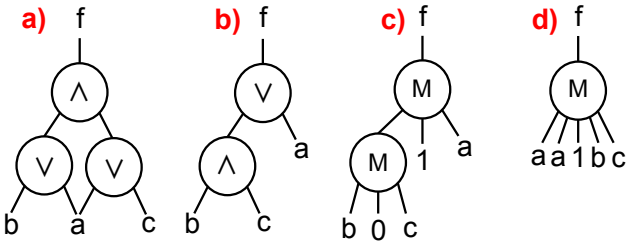

Fig. 1. Two-level representation example for the Boolean function $a \vee (b \wedge c)$ in forms: a) DNF, b) CNF, c) MNF and d) more compact MNF.

Example Boolean function $(a \wedge d') \vee (a \wedge b) \vee (a \wedge c) \vee (a' \wedge b \wedge c \wedge d')$. This form is already a DNF. A CNF is $(a \vee b) \wedge (a \vee c) \wedge (a \vee d') \wedge (b \vee c \vee d')$. A compact MNF is $M(a,a,b,c,d')$.

Example Boolean function $(a \wedge b \wedge c) \vee (a \wedge b \wedge d) \vee (a \wedge b \wedge e) \vee (a \wedge c \wedge d) \vee (a \wedge c \wedge e) \vee (a \wedge d \wedge e) \vee (b \wedge c \wedge d) \vee (b \wedge c \wedge e) \vee (b \wedge d \wedge e) \vee (c \wedge d \wedge e)$. This form is already a DNF.

A CNF can be obtained by just swapping $\wedge$ and $\vee$ operators. A compact MNF is $M(a, b, c, d, e)$.

Example Boolean function $a \oplus b \oplus c$. A DNF is $(a \wedge b \wedge c) \vee (a \wedge b' \wedge c') \vee (a' \wedge b \wedge c') \vee (a' \wedge b' \wedge c)$. A CNF is $(a' \vee b' \vee c) \wedge (a' \vee b \vee c') \wedge (a \vee b' \vee c') \wedge (a \vee b \vee c)$ A compact MNF is $M(a, M(a', b, c), M(a', b', c'))$.

Table I summarizes the sizes of the DNF, CNF and MNF encountered in the previous examples. The size of a CNF size is its number of clauses. Similarly, the size of a DNF is its number of terms. The size of an MNF is the number of majority operators appearing in the formula. As we can notice, the MNF is often more compact than CNF and DNF, with a size ranging from 1 to 4, while the corresponding CNF and DNF sizes range from 2 to 10. Similar results also emerged from theoretical studies on circuit complexity [8], [9]. Indeed, it has been shown in [8] that majority circuits of depth 2 and 3 possess the expressive power to represent arithmetic functions, such as powering, multiplication, division, addition etc., in polynomial size. On the other hand, CNF and DNF already require an exponential size for parity, majority and addition functions, which instead are polynomial with MNF [9].

TABLE I
TWO-LEVEL LOGIC REPRESENTATION COMPARISON.

Boolean Function	DNF	CNF	MNF
	Size	Size	Size
$a \vee (b \wedge c)$	2	2	1
$(a \vee b) \wedge (a \vee c) \wedge (a \vee d') \wedge (b \vee c \vee d')$	4	4	1
$(c' \vee b) \wedge (a' \vee c) \wedge (a \vee b)$	3	3	4
$M(a, b, c, d, e)$	10	10	1
$a \oplus b \oplus c$	4	4	3

So far, we have shown that two-level logic can be expressed in terms of majority operators in place of $\wedge$ and $\vee$. This comes at an advantage in representation size as compared to traditional CNF and DNF. Moreover, the natural properties of the majority function permit an uniform and efficient logic manipulation [2]. Still, further investigation and development of the topic are needed, as they will be discussed in Section VI. In the next section, we study the promising application of MNF formula to Boolean satisfiability.

IV. MAJORITY SATISFIABILITY

Boolean satisfiability, often abbreviated as SAT, is a core problem of computer science. New approaches to solve SAT, such as [10], [11], are of paramount interest to a wide class of computer applications. This is particularly relevant for *Electronic Design Automation* (EDA).

SAT is in general trivial for some representation form, such as DNF or *Binary Decision Diagrams* (BDDs) [12]. It is instead a difficult problem for CNF formula. For this reason, CNF-SAT is still actively studied. New SAT formulations are of great relevance when their representation can be derived from CNF in polynomial (preferably linear) time. The satisfiability of MNF formula falls in this category as MNF can be derived from CNF in linear time. This fact motivates us to study the general complexity of MNF-SAT.

A. Complexity of Unrestricted MNF-SAT

To classify the complexity of unrestricted MNF-SAT, we make use of a notable result from mathematical logic presented in [13]. Informally, the work in [13] investigates the satisfiability of a Boolean formula built with a finite set S of logic connectives, which is a problem referred to as S-SAT. For example, $(\wedge, 0)$-SAT is the satisfiability problem for a Boolean formula containing only $\wedge$ and constant 0 operators. It is demonstrated that a sufficient condition for S-SAT to NP-complete is the capability to represent the function $(x \wedge y')$ with only connectives drawn from S. If $(x \wedge y')$ is not representable by an S-formula, then S-SAT is decidable in polynomial time. The corresponding main theorem is reported verbatim from [13].

Theorem 4.1: (*Lewis' Theorem*) Let S be any finite set of connectives. Then S-SAT is NP-complete provided that there is an S-formula equivalent to $(x \wedge y')$. Otherwise S-SAT is decidable in polynomial time.

Proof The proof is given in [13].

To decide whether MNF-SAT is NP-complete or not, it is therefore sufficient to study its ability to express the function $(x \wedge y')$. This approach will also be used to classify the complexity of other SAT problems discussed in the rest of this paper. As stated, and proved, in the following theorem, MNF-SAT falls in the NP-complete complexity class.

Theorem 4.2: MNF-SAT is NP-complete.

Proof We can express $(x \wedge y')$ as $M(0, x, y')$, which is a valid MNF representation. Then, the NP-completeness follows from *Lewis' Theorem*.

Not surprisingly, MNF-SAT is as complex as CNF-SAT. Interestingly enough, alternative proofs, showing that MNF-SAT is a difficult problem, do exist. For example, one can notice that CNF-SAT (NP-complete [6]) can be reduced in polynomial time to MNF-SAT, that must then be also NP-complete [14].

B. Complexity of Some Restricted MNF-SAT

Even though MNF-SAT is in general a difficult problem, there are restrictions of MNF formula whose satisfiability can be determined easily. We define hereafter some MNF restrictions of interest.

Definition MNF_0 is an MNF where logic 1 is forbidden (also in the form of $0'$).

Example A valid MNF_0 is $M(M(a, b, 0), M(a, b', c), a)$. Instead, $M(M(a, b, 1), c', 0)$ is not an MNF_0 as logic 1 appears inside the formula.

Definition MNF_1 is an MNF where logic 0 is forbidden (also in the form of $1'$).

Example A valid MNF_1 is $M(M(a,1,d),M(a',b',e),1)$. Instead, $M(a,1,M(a',b,0))$ is not an MNF_1 as logic 0 appears inside the formula.

Definition MNF_{pure} is an MNF where both logic 1 and logic 0 are forbidden.

Example A valid MNF_{pure} is $M(M(a,b,c),M(a,b',c),a')$. If logic 1 or 0 appear in the MNF then it is not an MNF_{pure}.

Note that $\text{MNF}_0 \supset \text{MNF}_{pure}$ and $\text{MNF}_1 \supset \text{MNF}_{pure}$, but we keep them separated for the sake of reasoning.

Theorem 4.3: MNF_{pure}-SAT is always satisfiable.

Sketch of the Proof In [5], it is proven that a *self-dual* function fed with other *self-dual* functions remains *self-dual*. This is the case for MNF_{pure}, which is indeed always *self-dual*. A notable property of *self-dual* functions is to have an *on*-set of size 2^{n-1}, where n is the number of variables [5]. This means that an MNF_{pure} cannot reach an *on*-set of size 0 and therefore be unsatisfiable.

Informally, an MNF_0 is an MNF_{pure} with some inputs biased to logic 0.

Theorem 4.4: MNF_0-SAT is decidable in polynomial time.

Sketch of the Proof Let us assume that an MNF_0 is an MNF_{pure} where logic 0 is an additional variable, but always fixed to 0. In such an auxiliary MNF_{pure}, a greedy strategy can maximize the number of logic 1 to the final majority operator, just by setting the input variables to the most frequent polarity appearance. This can be done in linear time, assuming variables independence. With a real MNF_{pure}, this strategy would generate a satisfying input assignment. By recalling that our auxiliary MNF_{pure} has instead a variable fixed always to 0, the input pattern can now be unsatisfying. However, since we independently maximized the number of ones to the final majority, if such input assignment plus the fixed logic 0 cannot evaluate the MNF to 1 no other can do so. Consequently, MNF_0-SAT is decidable in polynomial time.

Note that there are different cases for which an MNF_0 can be unsatisfiable, such as $M(0,M(a,c,0),M(a',b,0))$.

Informally, an MNF_1 is an MNF_{pure} with some input biased to logic 1. As MNF_{pure} is always satisfiable, adding more logic 1 to the MNF cannot make it unsatisfiable. It follows that also MNF_1 is always satisfiable.

Corollary 4.5: MNF_1-SAT is always satisfiable.

Proof Suppose that by moving from MNF_{pure} to MNF_1 we can decrease the *on*-set of size from 2^{n-1} to 0, and therefore be unsatisfiable. The only additional element available to MNF_1, as compared to MNF_{pure}, is the logic 1. Clearly, an additional logic 1 is not possibly causing any decrease in monotone increasing functions. Hence, the *on*-set size cannot reach 0. It follows that MNF_1 formula are always satisfiable.

Whenever an MNF can be restricted to MNF_{pure} or MNF_1, its satisfiability is guaranteed, without need to check. If instead an MNF can be restricted only to MNF_0 its satisfiability can be determined efficiently, i.e., in polynomial time. We do not focus on algorithms to solve MNF_0-SAT, but we propose in the following section a general methodolody applicable to solve MNF-SAT.

V. Algorithm to Solve MNF-SAT

In order to automatically solve MNF-SAT instances, an algorithm is needed. We provide a core *decide* algorithm, with linear time complexity with respect to the MNF size. It exploits the intrinsic nature of MNF formula and can be embedded in a traditional *Decide - Deduce - Resolve* SAT solving approach [7]. We start from a one-level majority case and then we move to the two-level MNF case. Note that a recent work [19] considered the satisfiability of two-level (general) threshold circuits. It is proposed to reduce it to a *vector domination problem*. We differentiate from [19] by (i) focusing on MNF formula and (ii) developing a native solving methodology.

A. One-level Majority-SAT

In the case of a one-level majority function, the satisfiability check can be accomplished *exactly in linear time* by direct variable assignment (solely *decide* task). Informally, considering a single majority operator, a greedy strategy can maximize the number of logic 1 in an input pattern. If neither the pattern with the maximum number of logic 1 can evaluate a majority to 1, then no other input pattern can do so. An automated method for this task is depicted by Algorithm 1 and explained as follows. Each variable is processed in sequence, in any order. If the considered variable appears more often complemented than in its standard polarity, it is set to logic 0, otherwise to logic 1. At the end of this procedure, an assignment for the input variables to the majority operator

Algorithm 1 One-level Majority SAT

INPUT: Inputs x_1^n of a majority operator
OUTPUT: Assignment of x_1^n (if SAT this assignment evaluates to true, otherwise unSAT)

```
for (i=1; i≤n_vars; i++) do
  if x_i appears more often complemented then
    x_i = 0;
  else
    x_i = 1;
  end if
end for
if M(x_1^n) evaluates to 1 then
  return SAT;
else
  return unSAT;
end if
```

is obtained. If this assignment cannot evaluate the majority operator to true, then it is declared unsatisfiable, otherwise it is declared satisfiable. An example is provided hereafter.

Example The Boolean formula whose satisfiability we want to check is $M(a, b, a', a', b, c', c', d, e)$. To find an assignment which evaluates to logic 1, variables are considered in the order (a, b, c, d, e).

Variable a appears more often complemented in the majority operator, so it assigned to logic 0.

Variable b appears more often uncomplemented in the majority operator, so it assigned to logic 1.

Variable c appears more often complemented in the majority operator, so it assigned to logic 0.

Variable d appears more often uncomplemented in the majority operator, so it assigned to logic 1.

Variable e appears more often uncomplemented in the majority operator, so it assigned to logic 1.

The final assignment is then $(0, 1, 0, 1, 1)$ which evaluates $M(0, 1, 1, 1, 1, 1, 1, 1, 1) = 1$.

We have seen that the satisfiability of a single majority can be exactly decided in linear time, with respect to the size of the operator. The proposed greedy strategy is appropriate for such task. We show now how this procedure can be extended to handle two-level majority satisfiability.

B. Decide Strategy for MNF-SAT

For two-level MNF, a single *decide* may not be enough to determine SAT and it has to be iterated with *deduce* and *resolve* methods [7]. We propose here a *decide* strategy with linear time complexity with respect to the input MNF size. The rationale driving such process is to set each input variable to the logic value, 0 or 1, that maximizes the number of logic 1 in input to the final majority operator. A corresponding automated procedure is depicted by Algorithm 2 and explained as follows. A specific variable x_j is first passed to the procedure, together with the MNF structure information. Then, a metric is computed to decide the assignment of such variable to logic 0 or 1. The main difference with respect to the one-level majority is indeed the figure of merit used to drive the variable assignment. The description of a proper metric is as follows. Say n the number (odd) of inputs of the final majority in an MNF. Thus, there are n majorities in the MNF. Say m_i the number (odd) of inputs of the i-th majority operator, with $i \in \{1, 2, .., n\}$. Say $n_p(x_j, i)$ the number of occurence of variable x_j uncomplemented, in the i-th majority operator. Similarly, say $n_c(x_j, i)$ the number of occurence of variable x_j complemented, in the i-th majority operator. Using these informations, two cost metrics $C_p(x_j)$ and $C_n(x_j)$ are created. Such cost metrics range from 0 to 1 and indicate how much a positive (C_p) or negative (C_n) polarity assignment of a variable contribute to set the MNF to logic 1. They are computed as

$C_p(x_j) = (\sum_{i=1}^{n} n_p(x_j, i)/m_i)/n$ and
$C_n(x_j) = (\sum_{i=1}^{n} n_c(x_j, i)/m_i)/n$.

According to this rationale, if $C_p(x_j) < C_n(x_j)$ then variable x_j is assigned to logic 0, or assigned to logic 1 otherwise. At the end of this procedure, a valid assignment for variable x_j is obtained. If iterated over all the variables, Algorithm 2 determines a global assignment to evaluate the MNF. Such procedure can be used as core *decide* task in a traditional *Decide - Deduce - Resolve* SAT solving approach [7]. Note that also the *deduce* and *resolve* methods must be adapted to the MNF nature. Although, new and *ad hoc deduce* and *resolve* techniques are desirable, their study is out of the scope of the current paper. A simple example for the *decide* task, iterated over all the variables, is provided hereafter.

Algorithm 2 MNF-SAT *Decide* for a single variable

INPUT: Inputs: variable x_j, MNF structure
OUTPUT: Assignment for x_j most probably to SAT

```
compute n_p, n_c;
compute C_p(x_j), C_n(x_j);
if C_p(x_j) < C_n(x_j) then
    x_j = 0;
else
    x_j = 1;
end if
```

Example We want to determine the satisfiability for the MNF formula $M(M(a, b, c', d, 1), M(a, b', c', d, e'), M(a', b, 0))$. Variables are considered in the order (a, b, c, d, e) and their cost metrics are computed.

For variable a, $C_p(a) = (1/5 + 1/5 + 0/3)/3 = 0.13 > C_n(a) = (0/5 + 0/5 + 1/3)/3 = 0.11$, thus it is assigned to logic 1.

For variable b, $C_p(b) = (1/5 + 0/5 + 1/3)/3 = 0.17 > C_n(b) = (0/5 + 1/5 + 0/3)/3 = 0.06$, thus it is assigned to logic 1.

For variable c, $C_p(c) = (0/5+0/5+0/3)/3 = 0 < C_n(c) = (1/5 + 1/5 + 0/3)/3 = 0.13$, thus it is assigned to logic 0.

For variable d, $C_p(d) = (1/5 + 1/5 + 0/3)/3 = 0.13 > C_n(d) = (0/5+0/5+0/3)/3 = 0$, thus it is assigned to logic 1.

For variable e, $C_p(e) = (0/5 + 0/5 + 0/3)/3 = 0 < C_n(e) = (0/5 + 1/5 + 0/3)/3 = 0.06$, thus it is assigned to logic 0.

The obtained assignment is then $(1, 1, 0, 1, 0)$ which evaluates $M(M(1, 1, 1, 1, 1), M(1, 0, 1, 1, 1), M(0, 1, 0)) = M(1, 1, 0) = 1$. The initial MNF formula is already declared satisfiable.

Even though a single iteration may not be enough to determine the satisfiability of an MNF, the proposed *linear time decide* procedure can be used as core engine in a traditional SAT flow.

In the following section, we discuss the results obtained so far and highlight future research directions.

VI. Discussion and Future Work

Two-level logic representation and satisfiability are two linked problems that have been widely studied in the past

years. Nevertheless, the research in this field is still active. New approaches are continuously discovered and embedded in tools [10], [11], to push further the horizons of logic applications. The proposed MNF has the potential to enhance two-level logic representation and related SAT problems.

We demonstrated that any CNF or DNF can be translated in linear time into an MNF. However, in its unrestricted form, MNF leads to SAT problems as difficult as with CNF. Restricted versions of MNF exist, whose satisfiability can be decided in polynomial time. Advanced logic manipulation techniques capable to transform a general MNF into a restricted MNF can significantly simplify the MNF-SAT problem. Also, direct MNF construction from general logic circuits is of interest.

Regarding the MNF representation properties, it is still unclear whether a canonical form exists for MNF, as it does for CNF (product of maxterms) and DNF (sum of minterms). The discovery of a canonical MNF can reveal new promising features of majority logic.

In the context of MNF-SAT algorithms, a detailed study for MNF oriented *deduce* and *resolve* techniques is required. In this way, a complete MNF-SAT solver can be developed and its efficiency tested.

In summary, our next efforts are focused on (i) logic manipulation techniques for MNF, (ii) canonical MNF representation, (iii) MNF-oriented *deduce* and *resolve* techniques and (iv) development of an MNF-SAT tool.

VII. Conclusions

We presented, in this paper, an alternative two-level logic representation form based solely on majority and complementation operators. We called it *Majority Normal Form* (MNF). MNF is *universal* and potentially more compact than its CNF and DNF counterparts. Indeed, MNF includes both CNF and DNF representations. We studied the problem of MNF-*SATisfiability* (MNF-SAT) and we proved that it belongs to the NP-complete complexity class, as its CNF-SAT counterpart. However, we showed practical restrictions on MNF formula whose satisfiability can be decided in polynomial time. We have finally proposed a simple core procedure to solve MNF-SAT, based on the intrinsic functionality of two-level majority logic. Future research directions are oriented in the development of a complete MNF-SAT solver tool.

Acknowledgements

This research was supported by ERC-2009-AdG-246810.

References

[1] L. Amaru, P.-E. Gaillardon, G. De Micheli, *BDS-MAJ: A BDD-based logic synthesis tool exploiting majority logic decomposition*, Proc. DAC, 2013.

[2] L. Amaru, P.-E. Gaillardon, G. De Micheli, *Majority Inverter Graphs*, Proc. DAC, 2014.

[3] G. Yang, W. N.N. Hung, X. Song, M. Perkowski, *Majority-based reversible logic gates*, Theoretical Computer Science, 2005.

[4] H. Pospesel, *Introduction to Logic: Propositional Logic*, Pearson, 1999.

[5] T. Sasao, *Switching Theory for Logic Synthesis*, Springer, 1999.

[6] S. Cook, *The Complexity of Theorem-Proving Procedures*, Proc. ACM Symposium on Theory of Computing, 1971.

[7] A. Biere, M. Heule, H. van Maaren and T. Walsh *Handbook of Satisfiability*, IOS Press, 2009.

[8] M. Krause, P. Pudlak, *On the computational power of depth-2 circuits with threshold and modulo gates*, Theor. Comput. Sci., 174, pp. 137-156, 1997.

[9] AA. Sherstov, *Separating AC 0 from depth-2 majority circuits*, Proc. STOC, 2007.

[10] N. Een, N. Srensson, *MiniSat - A SAT Solver with Conflict-Clause Minimization*, SAT 2005.

[11] N. Een, A. Mishchenko, N. Sorensson, *Applying Logic Synthesis for Speeding Up SAT*, SAT 2007.

[12] R.E. Bryant, *Graph-based algorithms for Boolean function manipulation*, IEEE Transactions on Computers, C-35: 677-691, 1986.

[13] H. R. Lewis, *Satisfiability problems for propositional calculi*, Mathematical Systems Theory 13 (1979), pp. 45-53.

[14] M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

[15] K. Bernstein *et al.*, *Device and Architecture Outlook for Beyond CMOS Switches*, Proceedings of the IEEE, 98(12): 2169-2184, 2010.

[16] K. J. Chen, *et al.*, *InP-based high-performance logic elements using resonant-tunneling devices*, IEEE Electr. Dev. Lett., 17(3): 127-129, 1996.

[17] P. D. Tougaw, C. S. Lent, *Logical devices implemented using quantum cellular automata*, J. Applied Physics, 75(3): 1811-1817, 1994.

[18] M. De Marchi *et al.*, *Polarity control in Double-Gate, Gate-All-Around Vertically Stacked Silicon Nanowire FETs*, Proc. IEDM, 2012.

[19] R. Impagliazzo, *A Satisfiability Algorithm for Sparse Depth Two Threshold Circuits*, Arxiv 2013.