Abstract—In this paper, we establish a non-trivial duality between tautology and contradiction check to speed up circuit SAT. Tautology check determines if a logic circuit is true in every possible interpretation. Analogously, contradiction check determines if a logic circuit is false in every possible interpretation. A trivial transformation of a (tautology, contradiction) check problem into a (contradiction, tautology) check problem is the inversion of all outputs in a logic circuit. In this work, we show that exact logic inversion is not necessary. We give operator switching rules that selectively exchange tautologies with contradictions, and vice versa. Our approach collapses into logic inversion just for tautology and contradiction extreme points but generates non-complementary logic circuits in the other cases. This property enables computing benefits when an alternative, but equisolvable, instance of a problem is easier to solve than the original one. As a case study, we investigate the impact on SAT. There, our methodology generates a dual SAT instance solvable in parallel with the original one. This concept can be used on top of any other SAT approach and does not impose much overhead, except having to run two solvers instead of one, which is typically not a problem because multi-cores are widespread and computing resources are inexpensive. Experimental results show a 25% speed-up of SAT in a concurrent execution scenario. Also, statistical experiments confirmed that our runtime reduction is not of the random variation type.

I. INTRODUCTION

Inspecting the properties of logic circuits is pivotal to logic applications for computers and especially to Electronic Design Automation (EDA) [1]. There exists a large variety of properties to be checked in logic circuits, e.g., unateness, linearity, symmetry, balancedness, monotonicity, thresholdness and many others [2]. Basic characteristics are usually verified first to provide grounds for more involved tests. Tautology and contradiction are the most fundamental properties in logic circuits. A check for tautology determines if a logic circuit is true for all possible input patterns. Analogously, a check for contradiction determines if a logic circuit is false for all possible input patterns. While investigating elementary properties, tautology and contradiction check are difficult problems, i.e., co-NP-complete and NP-complete, respectively [3]. Indeed, both tautology and contradiction check are equivalent formulation of the Boolean SATisfiability (SAT) problem [3]. In this scenario, new efficient algorithms for tautology/contradiction check are key to push further the edge of computational limits, enabling larger logic circuits to be examined.

Tautology and contradiction check are dual problems. One can interchangeably check for tautology in place of contradiction by inverting all outputs in a logic circuit. In this trivial approach, the two obtained problems are fully complementary and there is no explicit computational advantage in solving one problem instead of the other.

In this paper, we show that exact logic inversion is not necessary for transforming tautology into contradiction, and vice versa. We give a set of operator switching rules that selectively exchange tautologies with contradictions. A logic circuit modified by our rules is inverted just if identically true or false for all input combinations. In the other cases, it is not necessarily the complement of the original one. In a simple logic circuit made of AND, OR and INV logic operators, our switching rules swap AND/OR operator types. We give a set of rules for general logic circuits in the rest of this paper. Note that in this paper we mostly deal with single output circuits. For multi-output circuits, the same approach can be extended by ORing (contradiction) or ANDing (tautology) the outputs that need to be checked into a single one.

Our approach generates two different, but equisolvable, instances of the same problem. In this scenario, solving both of them in parallel enables a positive computation speed-up. Indeed, the instance solved first stops the other reducing the runtime. This concept can be used on top of any other checking approach and does not impose much overhead, except having to run two solvers instead of one, which is typically not a problem because multi-cores are wide-spread and computing resources are inexpensive. Note that other parallel checking techniques exist. For example, one can launch in parallel many randomized check runs on the same problem instance with the aim to hit the instance-intrinsic minimum runtime [4]. Instead, in our methodology, we create a different but equi-checkable instance that has a potentially lower minimum runtime. As a case study, we investigate the impact of our approach on SAT. There, by using non-trivial and trivial dualities in sequence, we create a dual SAT instance solvable in parallel with the original one. Experimental results show 25% speed-up of SAT, on average, in a concurrent execution scenario. Also, statistical experiments confirmed that our runtime reduction is not of the random variation type.

The remainder of this paper is organized as follows. Section II describes some background and discusses the motivation for this study. Section III presents theoretical results useful for the scope of this paper. Section IV proves our main result on the duality between tautology and contradiction check. Section V shows the benefits enabled by this duality in SAT solving. Section VI concludes the paper.

II. BACKGROUND AND MOTIVATION

This section first provides notation on logic circuits. Then, it gives a brief background on tautology checking from an EDA perspective. Finally, it discusses the motivation for this study.

A. Notation

A logic circuit is a Directed Acyclic Graph (DAG) representing a Boolean function, with nodes corresponding to logic gates and directed edges corresponding to wires connecting the gates. The on-set of a logic circuit is the set of input patterns evaluating to true. Analogously, the off-set of a logic circuit is the set of input patterns evaluating to false.
logic gate is associated with a primitive Boolean function taken from a predefined set of basis logic operators, e.g., AND, OR, XOR, XNOR, INV, MAJ, MIN etc. Logic operators such as MAJ and MIN represent self dual Boolean functions, i.e., functions whose output complementation is equivalent to inputs complementation. A set of basis logic operators is said to be universal\(^1\) if any Boolean function can be represented by a logic circuit equipped with those logic gates. For example, the basis set \{OR, INV\} is universal while the basis set \{AND, MAJ\} is not. Fig.1 shows a logic circuit for the function \( f = \overline{(ab)d} + \overline{(ab)c} + dc \). The gates symbolic representation is shown in the box.

![Logic Circuit Example](image)

B. Tautology Checking

Tautology checking, i.e., verifying whether a logic circuit is true in every possible interpretation, is an important task in computer science and at the core of EDA [5], [7]. Traditionally, tautology checking supports digital design verification through combinational equivalence checking [7]. Indeed, the equivalence between two logic circuits can be detected by XNOR-ing and checking for tautology. Logic synthesis also uses tautology checking to (i) highlight logic simplifications during optimization [5], [6] and to (ii) identify matching during technology mapping [8]. On a general basis, many EDA tasks requiring automated deduction are solved by tautology check routines.

Unfortunately, solving a tautology check problem can be a difficult task. In its most general formulation, the tautology check problem is co-NP-complete. A straightforward method to detect a tautology is the exhasutive exploration of a function check problem is co-NP-complete. A straightforward method is as bad as any other tautology check method in the worst case but very efficient in real-life applications. With the rise of Binary Decision Diagrams (BDDs) [11], tautology check algorithms found an efficient canonical data structure explicitly showing the logic feature under investigation [12]. The BDD for a tautology is always a single node standing for the logic constant true. Hence, it is sufficient to build a BDD for a logic circuit and verify the resulting graph size (plus the output polarity) to solve a tautology check problem. Unfortunately, BDDs can be exponential in size for some functions (multipliers, hidden-weight bit, etc.). In the recent years, the advancements in SAT solving tools [13], [14] enabled more scalable approaches for tautology checking. Using the trivial duality between tautology and contradiction, SAT solvers can be used to determine if an inverted logic circuit is unsatisfiable (contradiction) and consequently if the original circuit is a tautology. Still, SAT solving is an NP-complete problem so checking for tautology with SAT is difficult in general.

C. Motivation

Tautology checking is a task surfing the edge of today’s computing capabilities. Due to its co-NP-completeness, tautology checking aggressively consumes computational power when the size of the problem increases. To push further the boundary of examinable logic circuits, it is important to study new efficient checking methodologies. Indeed, even a narrow theoretical improvement can generate a speed-up equivalent to several years of technology evolution.

In this paper, we present a non-trivial duality between contradiction and tautology check problems that opens up new efficient solving opportunities.

III. Properties of Logic Circuits

In this section, we show properties of logic circuits with regard to their on-set/off-set balance and distribution. These theoretical results will serve as grounds for proving our main claim in the next section.

We initially focus on two universal basis sets: \{AND, OR, INV\} and \{MAJ, INV\}. We deal with richer basis sets later on. We first recall a known fact about majority operators.

Property A MAJ operator of \(n\) variables, with \(n\) odd, can be configured as an \([n/2]\)-variables AND operator by biasing \([n/2]\) inputs to logic false and can be configured as an \([n/2]\)-variables OR operator by biasing \([n/2]\) inputs to logic true.

For the sake of clarity, an example of a three-input MAJ configuration in AND/OR is depicted by Fig. 2. Extended

![AND/OR Configuration](image)
Theorem 3.1: Logic circuits over the universal basis set \{MAJ, INV\}, with all inputs assignable (no logic constants), have \(|on-set|=2^{n-1}\) and \(|off-set|=2^{n-1}\), with \(n\) being the number of input variables.

Proof MAJ and INV logic operators, with no constants, represent self-dual Boolean functions. In \cite{2}, it is shown that self-dual Boolean functions have an \(|on-set|=|off-set|=2^{n-1}\), with \(n\) being the number of input variables. Also, it is shown in \cite{2} that Boolean functions composed by self-dual Boolean functions are self-dual as well. This is indeed the case for \{MAJ, INV\} logic circuits with no constants in input. As these circuits represent self-dual Boolean functions, we can assert \(|on-set|=|off-set|=2^{n-1}\). Q.E.D.

\{MAJ, INV\} logic circuits with no constants have a perfectly balanced partition between on-set size and off-set size. This is the case for the example in Fig. 3(d). Eventually, we know that by assigning \(d\) to true in such example circuit the on-set/off-set balance can be lost. Indeed, with \(d=true\) the \{MAJ, INV\} logic circuit then emulates the original \{AND, OR, INV\} logic circuit in Fig. 3(a), that could have different on-set size and off-set size. Still, it is possible to reclaim the perfect on-set/off-set balance by superposing the cases \(d=true\) and \(d=false\) in the \{MAJ, INV\} logic circuit. While we know precisely what the \{MAJ, INV\} logic circuit does when \(d=true\), the case \(d=false\) is not as evident. We can interpret the case \(d=false\) as an inversion in the MAJ configuration polarity. This means that where a MAJ is configured as an AND (OR) node in \(d=true\), it is instead configured as an OR (AND) node in \(d=false\). In other words, \(d=false\) in the \{MAJ, INV\} logic circuit of Fig. 3(d) corresponds to switch AND/OR operator types in the original \{AND, OR, INV\} logic circuit of Fig. 3(a). The resulting AND/OR switched circuit is depicted by Fig. 3(c).

United by a common \{MAJ, INV\} generalization, \{AND, OR, INV\} logic circuits and their AND/OR switched versions share strong properties about on-set/off-set reparation. The following theorem states their relation.

Theorem 3.2: Let \(A\) be a logic circuit over the universal basis set \{AND, OR, INV\}. Let \(A'\) be a modified version of \(A\), with AND/OR operators switched. The following identities hold \(|on-set(A)|=|off-set(A')|\) and \(|off-set(A)|=|on-set(A')|\).

Proof Say \(M\) a \{MAJ, INV\} logic circuit emulating \(A\) using an extra fictitious input variable, say \(d\). \(M_{d=0}\) is structurally and functionally equivalent to \(A\), while \(M_{d=0}\) is structurally and functionally equivalent to \(A'\). From Theorem 3.1 we know that \(|on-set(M)|=|off-set(M)|=2^{n-1}=2^{m}\), where \(m\) is the number of input variables in \(A\) and \(n\) the number of input variables in \(M\), with \(n=m+1\) to take into account the extra fictitious input variable in \(M\). We know by construction that \(|on-set(M_{d=1})|+|on-set(M_{d=0})|=2^{n-1}=2^{m}\) and \(|off-set(M_{d=1})|+|off-set(M_{d=0})|=2^{n-1}=2^{m}\). Again by construction we know that \(M_{d=1}\) and \(M_{d=0}\) can be substituted by \(A\) and \(A'\), respectively, in all equations. Owing to the basic definition of \(A\) and \(A'\) we have that \(|on-set(A)|+|off-set(A)|=2^{m}\) and \(|on-set(A')|+|off-set(A')|=2^{m}\). Expressing \(|on-set(A)|\) as \(2^{m-|on-set(A')|}\) from the first set of equations and substituting this term in \(|on-set(A)|+|off-set(A)|=2^{m}\) we get \(2^{m-|on-set(A')|}+|off-set(A)|=2^{m}\) that can be simplified as \(|off-set(A)|=|on-set(A')|\). This proves the first identity of the Theorem. The second identity can be proved analogously.
Q.E.D.

Informally, the previous theorem says that by switching AND/OR operators in an \{AND, OR, INV\} logic circuit we swap the on-set and off-set sizes. From a statistical perspective, this is equivalent to invert \(Pr(A=true)\) with \(Pr(A=false)\), under uniformly random input string of bits. While this also happens with exact logic inversion, here the actual distribution of the on-set/off-set elements is not necessarily complementary. In the next section, we show the implications of the theoretical results seen so far in tautology and contradiction check problems.

IV. FROM TAUTOLOGY TO CONTRADICTION AND BACK

Verifying whether a logic circuit is a tautology, a contradiction or a contingency is an important task in logic applications for computers.

In this section, we show that tautology and contradiction check in logic circuits are dual and interchangeable problems that do not require exact logic inversion per se. We start by considering logic circuit over the universal basis set \{AND, OR, INV\} and we consider richer basis sets later on. The following theorem describes the non-trivial duality between tautology and contradiction in \{AND, OR, INV\} logic circuits.

**Theorem 4.1:** Let \(A\) be a logic circuit over the universal basis set \{AND, OR, INV\} representing a tautology (contradiction). The logic circuit \(A'\), obtained by switching AND/OR operations in \(A\), represents a contradiction (tautology).

**Proof** If \(A\) represents a tautology then \(|on-set(A)|=2^m\) and \(|on-set(A)|=0\), with \(m\) being the number of inputs. Owing to Theorem 3.2 \(|on-set(A')|=|off-set(A)|=0\) and \(|off-set(A')|=|on-set(A)|=2^m\). It follows that \(A'\) is a contradiction. Analogous reasoning holds for contradiction to tautology transformation. Q.E.D.

Switching AND/ORs in an \{AND, OR, INV\} logic circuit is strictly equivalent to logic inversion only for tautology and contradiction. In the other cases, \(A\) and \(A'\) are not necessarily complementary. We give empirical evidences about this fact hereafter. Fig. 4 depicts the obtained results in a graph chart. We examined 17 random Boolean functions of four input variables, with on-set size ranging from 0 (contradiction) to 16 (tautology). We first compared the on-set size of the real inverted logic circuits with the on-set size of the AND/OR switched circuits. As expected, Theorem 3.2 holds and switching AND/OR operators results in exchanging the on-set and off-set sizes. This also happens with the real inverted circuits, but in that case also the actual on-set/off-set elements distribution is complementary. To verify what is the on-set/off-set elements distribution in general, we define a distance metric between the real inverted and AND/OR switched circuits. The distance metric is computed in two steps. First, the truth tables of the circuits are unrolled, using the same input order, and represented as binary strings. Second, the distance metric is measured as the Hamming distance between those binary strings, of equal size, is the number of positions at which the corresponding bits are different.

Recalling the example in Fig. 3(a), the original logic circuit represents a tautology. Consequently, the logic circuit in Fig. 3(c) represents a contradiction. These properties are verifiable by hand as the circuits considered are small. For an example which is a contingency, consider the \{AND, OR, INV\} circuit realization for \(f = ab' + c'\) (contingency). By switching AND/ORs, we get \(g = (a + b)c'\) which is different from both \(f\) or \(f'\), as anticipated.

Fig. 4: Comparison between real inverted and AND/OR switched logic circuits representing 4-variable Boolean functions. The on-set size ranges from 0 to \(2^4\).
TABLE I: Switching Rules for Tautology/Contradiction Check

<table>
<thead>
<tr>
<th>Original Logic Operator</th>
<th>Switched Logic Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>INV</td>
<td>INV</td>
</tr>
<tr>
<td>AND</td>
<td>OR</td>
</tr>
<tr>
<td>OR</td>
<td>AND</td>
</tr>
<tr>
<td>MAJ</td>
<td>MAJ</td>
</tr>
<tr>
<td>XOR</td>
<td>XNOR</td>
</tr>
<tr>
<td>XNOR</td>
<td>XOR</td>
</tr>
</tbody>
</table>

switching logic operators in $A$ as per Table I, represents a contradiction (tautology).

**Proof** In order to prove the theorem, we need to show the switching rules just for XOR, XNOR and MAJ operators. AND/OR switching is already proved by Theorem 4.1. Consider the XOR operator decomposed in terms of \{AND, OR, INV\}: $f = a \oplus b = ab' + a'b$. Applying the duality in Theorem 4.1 we get $g = (a + b')(a' + b)$ that is indeed equivalent to a XNOR operator. This proves the XOR to XNOR switching and vice versa. Analogously, consider the MAJ operator decomposed in terms of \{AND, OR, INV\}: $f = ab + ac + bc$. Applying the duality in Theorem 4.1 we get $g = (a + b)(a + c)(b + c)$ that is still equivalent to a MAJ operator. Hence, MAJ operators do not need to be modified. Q.E.D.

Note that in a data structure for a computer program, the operator switching task does not require actual pre-processing of the logic circuit. Indeed, each time that a node in the DAG is evaluated an external flag word determines if the regular or switched operator type has to be retrieved from memory.

In the current section, we shown a non-trivial duality between contradiction and tautology check. In the next section, we study its application on Boolean satisfiability.

V. EXPERIMENTAL RESULTS

In this section, we exercise our non-trivial duality in Boolean SATisfiability (SAT) problems. First, we describe how to use the tautology/contradiction duality to generate a second (dual) equisatisfiable SAT instance. Second, we demonstrate that the dual instance can be solved faster than the regular one and the corresponding runtime reduction is not of the random variation type. Third, and last, we show experimental results for a concurrent regular/dual SAT execution scenario.

A. Boolean SAT and Tautology/Contradiction Duality

The Boolean SAT problem consists of determining whether there exists or not an interpretation evaluating to true a Boolean formula or circuit. The Boolean SAT problem is reciprocal to a check for contradiction. When contradiction check fails then Boolean SAT succeeds while when contradiction check succeeds then Boolean SAT fails. Instead of checking for Boolean SAT or for contradiction, one can use a dual transformation in the circuit and check for tautology. Such transformation can be either (i) non-trivial, i.e., switching logic operators in the circuit as per Table I or (ii) trivial, i.e., output complementation. If we use twice any dual transformation, we go back to the original problem domain (contradiction, SAT). Note that if we use twice the same dual transformation (trivial-trivial or non-trivial-non-trivial) we obtain back exactly the original circuit. Instead, if we apply two different dual transformations in sequence (trivial-non-trivial or non-trivial-trivial) we obtain an equisatisfiable but not necessarily equivalent circuit. We use the latter approach to generate a second equisatisfiable circuit, which we call the dual circuit. The dual circuit SAT can be solved in parallel with the regular one in a “first finishing wins” speculative strategy. Fig. 5 depicts the corresponding flow. We generate the dual circuit by first applying our non-trivial duality (switching rules in Table I) and finally complementing the outputs (trivial duality). Note that these operations ideally require no (or very little) computational overhead, as explained previously.

B. Verification of SAT Solving Advantage on the Dual Circuit

In our first set of experiments we focused on verifying that the dual circuit can be easier to satisfy than the regular circuit. For this purpose, we modified MiniSat-C v1.14.1 [16] to read circuits in AIGER format [18] and to encode them in CNF internally via Tseitin transformation. The dual circuit is generated online during reading if a switch “-p” is given. We considered a large circuit (0.7 M nodes) over 1000 randomized (pseudo-random number generator seed) runs. Fig. 6 shows the runtime distributions for dual and regular SAT. The dual runtime distribution is clearly left-shifted (but partially overlapping) with respect to the regular runtime distribution. This confirms that (i) the dual circuit can be solved faster than the regular one and (ii) the runtime reduction is not of the random variation type.

C. Results for Concurrent Regular/Dual SAT Execution

In our second set of experiments (downloadable at [19]) we used ABC tool [17] to test our dual approach together...
with advanced techniques to speed-up SAT. Our custom set of benchmarks is derived by (i) unfolding SAT sequential problems (ii) encoding combinational equivalence check problems. All benchmarks are initially described in Verilog as a netlist of logic gates over the basis \{AND, OR, INV, XOR, XNOR, MAJ\}. The dual circuits are obtained by applying switching rules in Table I and inverting the output. The ABC script to read and run SAT on these benchmarks is: `read library,genlib; r-m input.v; s; write out.aig; &r out.aig; &write cnf -K 4 out.cnf; dsat -p out.cnf;` Apart from standard I/O commands, note that `&write cnf -K 4 out.cnf` generates a CNF using a technology mapping procedure and `dsat -p` calls MiniSat with variable polarity alignment.

Table II shows results for regular vs. dual SAT solving with our setup. For about half of the benchmarks (7/13) the dual instance concluded first while for the remaining ones (6/13) the regular instance was faster. The total regular runtime is quite close to the total dual runtime (just 6% of deviation). However, considering here the speculative parallel SAT flow in Fig. 5, we can ideally reduce the total runtime by about 25%. Note that this is an ideal projection into a parallel execution environment, with no overhead. We experimentally verified that the average overhead can be small (few percentage points) thanks to the intrinsic independence of the two tasks.

TABLE II: Experimental Results for Regular vs. Dual SAT Solving

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>I/O</th>
<th>Logic Size</th>
<th>Logic Depth</th>
<th>Runtime Regular</th>
<th>Runtime Dual</th>
<th>[Δ Runtime]</th>
<th>Best Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>hardsat1</td>
<td>458/1</td>
<td>28.35</td>
<td>22</td>
<td>186.35</td>
<td>127.35</td>
<td>58.9</td>
<td></td>
</tr>
<tr>
<td>hardsat2</td>
<td>458/1</td>
<td>28.63</td>
<td>22</td>
<td>191.87</td>
<td>140.77</td>
<td>51.1</td>
<td></td>
</tr>
<tr>
<td>hardsat3</td>
<td>57/27</td>
<td>36.96</td>
<td>26</td>
<td>9.1</td>
<td>1</td>
<td>0.64</td>
<td></td>
</tr>
<tr>
<td>hardsat4</td>
<td>37/25/1</td>
<td>45.12</td>
<td>30</td>
<td>68.82</td>
<td>20.53</td>
<td>48.29</td>
<td></td>
</tr>
<tr>
<td>hardsat5</td>
<td>37/25/1</td>
<td>45.12</td>
<td>40</td>
<td>40.91</td>
<td>23.72</td>
<td>17.19</td>
<td></td>
</tr>
<tr>
<td>hardsat6</td>
<td>30/65/1</td>
<td>86.08</td>
<td>43</td>
<td>37.51</td>
<td>64.24</td>
<td>26.73</td>
<td></td>
</tr>
<tr>
<td>hardsat7</td>
<td>37/22/40/1</td>
<td>35.96</td>
<td>51</td>
<td>4.8</td>
<td>1.12</td>
<td>3.68</td>
<td></td>
</tr>
<tr>
<td>Total sat</td>
<td>19118/27/1</td>
<td>25977/1</td>
<td>25.39</td>
<td>390.43</td>
<td>363.04</td>
<td>27.39</td>
<td>195.38</td>
</tr>
</tbody>
</table>

Note that this is an ideal projection into a parallel execution scenario.

**ACKNOWLEDGMENTS**

This work was supported by grant ERC-2009-AdG-246810.

**REFERENCES**

[16] MiniSat SAT solver available online at http://minisat.se/MiniSat.html
[18] AIGER benchmarks available online at http://fmv.jku.at/aiger/