ISSN 1870-4069

Applying an Incremental Satisfiability Algorithm to Automatic Test Pattern Generation

Guillermo De Ita, Meliza Contreras, Pedro Bello

Benemrita Universidad Autnoma de Puebla,
Faculty of Computer Sciences, Puebla, Mexico

deita@cs.buap.mx, mcontreras@cs.buap.mx,
pbello@cs.buap.mx

Abstract. We propose a novel method to review the satisifiability of $(K \wedge \phi)$, where K is a two conjunctive form and ϕ is a three conjunctive form, both formulas defined on the same set of variables. We extend our method to solve the incremental satisfiablity problem (ISAT), and we present different cases where ISAT can be solved in polynomial time. Our proposal is adequate to solve the 2-ISAT problem, and our method allows to recognize tractable instances of 2-ISAT. We illustrate a practical application of our algorithm in the area of recognizing faults on combinatorial circuits.

Keywords. Satisfiability problem, incremental satisfiability problem, 2-SAT, propositional entailment problem, efficient satisfiability instances.

1 Introduction

A central issue in determining these frontiers has centered in the satisfiability problem (SAT) in the propositional calculus [3]. The case 2-SAT, that determines the satisfiability of propositional two Conjunctive Normal Forms (2-CF), is an important tractable case of SAT.

SAT is an important theoretical problem since it was proved as the first problem in the NP-complete complexity class. Despite the theoretical hardness of SAT, current state-of-the-art decision procedures for SAT, known as SAT solvers, have become surprisingly efficient. Subsequently these solvers have found many industrial applications. Such applications are rarely limited to solving just one decision problem, instead, a single application will typically solve a sequence of related problems. Modern SAT solvers handle such problem sequences as an instance of the incremental satisfiability problem (ISAT) [10].

We will consider the ISAT problem as a dynamic incremental set of clauses: $F_0, F_1, \ldots, F_n$, starting with an initial satisfiable formula F_0. Each F_i results from a change in the preceding one F_{i-1} imposed by the 'outside world'. Although the change can be a restriction (add clauses) or a relaxation (remove clauses), we will focus in the restriction case, so we consider adding a new set of clauses to F_{i-1} in order to form F_i. The process of adding new clauses is finished when F_i is unsatisfiable or there are no more clauses to be added.

One idea used on ISAT methods, is to preserve the structures formed when previous formulas were processed, allowing the recognition of common subformulas that they were previously considered. More importantly, it allows the solver to reuse information across several related consecutive problems. The resulting performance improvements make ISAT a crucial feature for modern SAT solvers in real-life applications [10].

ISAT is of interest to a large variety of applications that need to be processed in an evolutive environment [3]. This could be the case of applications such as reactive scheduling and planning, dynamic combinatorial optimization, reviewing faults in combinatorial circuits, dynamic constraint satisfaction and machine learning in a dynamic environment [9].

In [3], we designed an algortihm for reviewing $Sat(K \wedge \phi)$, K and ϕ being CF's. In this work, we adapt our initial algorithm considering that K is a 2-CF and ϕ a 3-CF. We present here, a study about the threshold for the 2-ISAT problem that could be relevant to understand the border between P and NP complexity classes. We also show the practical relevance of our algorithm in the area of automatic test pattern generation (ATPG) systems that consists in differentiating defective components from defect-free components.

2 Preliminaries

Let $X = \{x_1, \ldots, x_n\}$ be a set of n *Boolean variables*. A *literal* is either a variable x_i or a negated variable $\overline{x}_i$. As usual, for each $x \in X$, $x^0 = \neg x = \overline{x}$ and $x^1 = x$.

A *clause* is a disjunction of different and non-complementary literals. Notice that we discard the case of tautological clauses. For $k \in I\!N$, a k-*clause* is a clause consisting of exactly k literals, and a $(\leq k)$-clause is a clause with at most k literals.

A *conjunctive normal form* (CNF, or just CF) F is a conjunction of non-tautological clauses. We say that F is a monotone positive CF if all of its variables appear in unnegated form. A k-CF is a CF containing only k-clauses. $(\leq\ k)$-CF denotes a CF containing clauses with at most k literals.

A variable $x \in X$ *appears* in a formula F if either x or $\neg x$ is an element of F. The *size* of a CF F is defined as the total number of literals appearing in the CF F. We use $\upsilon(X)$ to represent the variables involved in the object X, where X can be a literal, a clause, or a CF. For instance, for the clause $c = \{\overline{x_1}, x_2\}$, $\upsilon(c) = \{x_1, x_2\}$. $Lit(F)$ is the set of literals involved in F, i.e. if $X = \upsilon(F)$, then $Lit(F) = X \cup \overline{X} = \{x_1, \overline{x}_1, ..., x_n, \overline{x}_n\}$. Also, we used $\neg Y$ as the negation operator on the object Y.

An *assignment* s for F is a function $s : \upsilon(F) \rightarrow \{0, 1\}$. An *assignment* s can also be considered as a set of literals without a complementary pair of literals, e.g., if $l \in s$, then $\overline{l} \notin s$, in other words s turns l *true* and $\overline{l}$ *false* or viceversa. Let c be a clause and s an assignment, c is *satisfied* by s if and only if $c \cap s \neq \emptyset$. On the other hand, if for all $l \in c$, $\overline{l} \in s$, then s falsifies c.

Let F be a CF, F is *satisfied* by an assignment s if each clause in F is satisfied by s. F is *contradicted* by s if any clause in F is falsified by s. A model of F is an assignment for $\upsilon(F)$ that satisfies F. A falsifying assignment of F is an assignment for $\upsilon(F)$ that contradicts F. If $n = |\upsilon(F)|$, then there are 2^n possible assignments defined over $\upsilon(F)$. Let $S(F)$ be the set of 2^n assignments defined over $\upsilon(F)$. $s \vdash F$

denotes that assignment s is a model of F, while that $s \not\models F$ denotes that s is a falsifying assignment of F.

If $F_1 \subset F$ is a formula consisting of some clauses from F, and $\upsilon(F_1) \subset \upsilon(F)$, an assignment over $\upsilon(F_1)$ is a *partial* assignment over $\upsilon(F)$. If $n = |\upsilon(F)|$ and $n_1 = |\upsilon(F_1)|$, any assignment over $\upsilon(F_1)$ has 2^{n-n_1} extensions as assignments over $\upsilon(F)$. If s has logical values determined for all variables in F then s is a *total assignment* of F.

The SAT problem consists of determining whether F has a model. SAT(F) denotes the set of models of F, then SAT(F) $\subseteq S(F)$. The set FAL(F) $= S(F) \setminus SAT(F)$ consists of the assignments from $S(F)$ that falsify F. Clearly, for any propositional formula F, $S(F) = SAT(F) \cup Fals(F)$.

3 Reducing Conjunctive Normal Forms

Let K be a CF, i.e., $K = \bigwedge_{i=1}^{m} C_i$, where each $C_i, i = 1, \ldots, m$ is a disjunction of literals. Let us introduce the main problem to be considered here.
Instance: Let K be a 2-CF and ϕ be a 3-CF, such that $\upsilon(\phi) \subseteq \upsilon(K)$.
Problem: To determine SAT($K \cup \phi$).

We will present here, an efficient algorithm to solve this problem. But first, we introduce some common rules used to simplify a conjunctive normal form F, keeping just the necessary subformulas that determine the satisifiability of F. For example, for a CF it is common to delete all redundant clauses as: tautological clauses, repeated clauses and clauses with pure literals.

Subsumed clause Rule: Given two clauses c_i and c_j of a CF F, if $Lit(c_i) \subseteq Lit(c_j)$ then c_j is subsumed by c_i, and c_j can be deleted from F, because all satisfying assignment of c_j is a satisfying assignment of c_i, that is $Sat(c_j) \subseteq Sat(c_i)$. Thus, it is enough just to keep c_i (the clause which subsumes) in the CF.

Furthermore, subsumed clause rule can be combined with resolution in order to simplify ϕ, as we show in the following lemma.

Lemma 1. *Let $(x, y) \in K$ and a clause $(\neg x, y, z) \in \phi$ then its resolvent (y, z) is a binary clause that can be added to K and its father $(\neg x, y, z)$ can be deleted from ϕ.*

Proof. We have that $(x \vee y) \wedge (\neg x \vee y \vee z) \equiv y \vee (x \wedge (\neg x \vee z)) \equiv y \vee ((x \wedge \neg x) \vee (x \wedge z)) \equiv y \vee (x \wedge z) \equiv (x \vee y) \wedge (y \vee z)$, and if these last two clauses are preserved in K then the clause $(\neg x, y, z)$ can be deleted from ϕ, because it is subsumed by $(y \vee z) \in K$ and therefore, the set of models of $(K \wedge \phi)$ are preserved without changes.

Resolution is also useful in our purpose to move clauses from ϕ to K. For example, if ϕ contains clauses type: (x, y, z) and $(\neg x, y, z)$, then they can be deleted from ϕ and the clause (y, z) is added to K. The justification of this rule comes from the distributive property, since $(x \vee y \vee z) \wedge (\neg x \vee y \vee z) \equiv (y \vee z) \vee (x \wedge \neg x) \equiv (y \vee z)$.

On the other hand, other common rules used to simplify formulas have to be adapted for our purpose.

Rule of Pure Literal: Let F be a CF, $l \in Lit(F)$ is a pure literal if l appears in F but $\overline{l}$ does not appear in F.

If a clause contains a pure literal, that clause can be eliminated from F, keeping the logical value of F. Because if the literal l is set to $True$, the clause containing l is also $True$, therefore it can be deleted from F. However, this rule has to be applied carefully for our problem, since in the process of adding new clauses ϕ to K, the initial pure literals in K could be not longer pure in $K \cup \phi$. Therefore, to delete clauses with pure literals must be applied into a local reach, working in each instance $(K \wedge \phi)$. But, if a new set of clauses ϕ_{i+1} has to be considered, then all clause with pure literals deleted from $(K \wedge \phi)$ must be returned to ϕ_{i+1}.

It is easy to build $Fals(K)$ since each clause C_i determines a subset of falsifying assignments of K. For example, $Fals(K) = \bigcup_{i=1}^{m} Fals(C_i)$. The following lemma expresses how to form the falsifying set of assignments of a CF.

Lemma 2. *Let $K = \bigwedge_{i=1}^{m} C_i$ be a CF, then $Fals(K) = \bigcup_{i=1}^{m} \{\sigma \in S(K) \mid Fals(C_i) \subseteq \sigma\}$*

Lemma 3. *If a CF K is satisfiable, then $\forall K' \subseteq K$, K' is a CF satisfiable.*

Proof. If K is satisfiable, then $Fals(K) = \bigcup_{C_i \in K} Fals(C_i) \subset S(F)$. Clearly, if we discard some clauses from K, forming K', then $Fals(K') = \bigcup_{C_i \in K'} Fals(C_i) \subseteq \bigcup_{C_i \in K} Fals(C_i) \subset S(F)$. Thus, K' is satisfiable.

Corollary 1 *If a CF K is unsatisfiable, then $\forall$ CF K' such that $K \subseteq K'$, K' remains unsatisfiable.*

Proof. An unsatisfiable CF K holds that $Fals(K) = \bigcup_{C_i \in K} Fals(C_i) = S(F)$. Then, if we aggregate more clauses to K forming K', then $Fals(K) = \bigcup_{C_i \in K} Fals(C_i) \subseteq \bigcup_{C_i \in K'} Fals(C_i) = S(F)$. Thus, K' is also unsatisfiable.

4 The Transitive Closure of a 2-CF

The fact that in a 2-CF formula a clause is equivalent to a pair of implications can be straightforward established as follows: if $\{x, y\} \in F$ then $\{x, y\}$ is equivalent to both $\overline{x} \rightarrow y$ and $\overline{y} \rightarrow x$. The arrow $\rightarrow$ has the usual meaning of implication in classical logic.

Definition 1 *Let F be a 2-CF and L its set of literals. The relation $\rightarrow_R \subset L \times L$ is defined as follows: $x \rightarrow_R y$ if and only if $x \rightarrow y$.*

Definition 2 *Let F be a 2-CF, a partial assignment s of F is a feasible model for F, if s does not falsify any clause in F.*

We consider now the transitive closure of $\rightarrow_R$, denoted by "$\Rightarrow$". This new relation $\Rightarrow$ can always be constructed inductively from $\rightarrow_R$. For any feasible model s of F where x and y occur in F; if $x \Rightarrow y$ and x is true in s then it is straightforward to show

that y is true in s. It is said that y is *forced* to be true by x. Let $T(x)$ be the set of literals forced to be true by x, that is $T(x) = \{x\} \cup \{y : x \Rightarrow y\}$.

It is clear that, if x is a literal occurring in a formula F, and if $\bar{x} \in T(x)$, then x cannot be set to true in any model of F. Analogously, if $x \in T(\bar{x})$ then x cannot be set to false in any model of F.

Definition 3 *Let F be a 2-CF, for any literal $x \in F$, it is said that $T(x)$ is inconsistent if $\overline{x} \in T(x)$ or $\perp \in T(x)$, otherwise $T(x)$ is said to be consistent.*

Unit clauses in 2-CF can be expressed as implications, that is, if F has unit clauses $\{u\}$ then $u \equiv u \vee \perp$, hence $\perp \in T(\overline{u})$. As a consequence, in formulas with unit clause $\{u\}$ follows that $T(\overline{u})$ is inconsistent. Let F be a 2-CF with n variables and m clauses, it has been shown that for any literal $x \in F$, $T(x)$ and $T(\overline{x})$ are computed in polynomial time over $|F|$, in fact, for all $l \in Lit(F)$, $T(l)$ is computed with time complexity $O(n \cdot m)$ [5].

For any literal x in a 2-CF, the sets $T(x)$ and $T(\overline{x})$ allow to determine which variables have a fixed logical values in every model of F, that is to say, the variables that are true in every model of F and the variables that are false in every model of F. The properties of the sets $T(x)$ and $T(\overline{x})$ will be established as a lemma.

Lemma 4. *Let F be a 2-CF and x a variable in F.*

1. *If $T(x)$ is inconsistent and $T(\overline{x})$ is consistent then $\overline{x}$ is true in every model of F.*
2. *If $T(\overline{x})$ is inconsistent and $T(x)$ is consistent then x is true in every model of F.*
3. *If both $T(\overline{x})$ and $T(x)$ are inconsistent then F does not have models and F is unsatisifiable.*
4. *If both $T(\overline{x})$ and $T(x)$ are consistent then x does not have a fixed valued in each model of F.*

Proof. 1. Suppose $\overline{x}$ is false in a model of F, so x should be true in that model of F. However, $T(x)$ is inconsistent, so $x \Rightarrow \overline{x}$ and x cannot be true in the model of F contradicting the assumption. Hence, any model of F has to assign false to x and true to $\overline{x}$. The other cases are proved similarly.

From properties (1) and (2) of lemma 4 we formulate the following definition

Definition 4 *A base for the set of models of a 2-CF F, denoted as $S(F)$, is a partial assignment s of F which consists of the variables with a fixed truth value.*

We denote by *Transitive_Closure(F)* to the procedure which computes the sets $T(x)$ and $T(\bar{x})$ for each $x \in \upsilon(F)$. The transitive procedure applied on a 2-CF F allows to build bases for the set of models of F. If a base $S(F)$ is such that $|S(F)| = |\upsilon(F)|$, then each variable of F has a fixed truth value in every model of F, so there is just one model.

Definition 5 *Let F be a 2-CF and x a literal of F. The reduction of F by x, also called $forcing\ x$ and denoted by $F[x]$, is the formula generated from F by the following two rules*

a) removing from F the clauses containing x (subsumption rule),
b) removing $\overline{x}$ from the remaining clauses (unit resolution rule).

A reduction is also sometimes called a *unit reduction*. The reduction by a set of literals can be inductively established as follows: let $s = \{l_1, l_2, \ldots, l_k\}$ be a partial assignment of $v(F)$. The reduction of F by s is defined by successively applying definition 5 for l_i, $i = 1, \ldots, k$. That is reduction of F by l_1 gives the formula $F[l_1]$, following a reduction of $F[l_1]$ by l_2, giving as a result the formula $F[l_1, l_2]$ and so on. The process continues until $F[s] = F[l_1, ..., l_k]$ is reached. In case that $s = \emptyset$ then $F[s] = F$.

Example 1. Let $F = \{\{x_1, \overline{x}_2\}, \{x_1, x_2\}, \{x_1, x_3\}, \{\overline{x}_1, x_3\}, \{\overline{x}_2, x_4\}, \{\overline{x}_2, \overline{x}_4\}, \{x_2, x_5\}, \{x_3, \overline{x}_5\}\}$. If $s = \{x_2, \overline{x}_3\}$, $F[x_2] = \{\{x_1\}, \{x_1, x_3\}, \{\overline{x}_1, x_3\}, \{x_4\}, \{\overline{x}_4\}, \{x_3, \overline{x}_5\}\}$, and $F[s] = \{\{x_1\}, \{x_1\}, \{\overline{x}_1\}, \{x_4\}, \{\overline{x}_4\}, \{\overline{x}_5\}\}$.

Let F be a 2-CF formula and s a partial assignment of F. If a pair of contradictory unitary clauses is obtained while $F[s]$ is being computed, then F is falsified by the assignment s. Furthermore, during the computation of $F[s]$, new unitary clauses can be generated. Thus, the partial assignment s is extended by adding the already found unitary clauses, that is, $s = s \cup \{u\}$ where $\{u\}$ is a unitary clause. So, $F[s]$ can be again reduced using the new unitary clauses. The above iterative process is generalized, and we call to this iterative process $Unit\_Propagation(F, s)$. For simplicity, we will abbreviate $Unit\_Propagation(F, s)$ as $UP(F, s)$.

As a result of applying $UP(F, s)$, we obtain a new assignment s' that extend to s, and a new subformula F' formed by the clauses from F that are not satisfied by s'. We denote $(F', s') = UP(F, s)$ to the pair resulting of the application of Unit Propagation on F by the assignment s. Notice that if s falsifies F then s' could have complementary literals and F' contains the null clause. And when s satisfies F, then F' is empty.

5 Incremental Satisfiability Problem

The incremental satisfiability problem (ISAT) involves checking whether satisfiability is maintained when new clauses are added to an initial satisfiable knowledge base K. ISAT is considered as a generalization of SAT since it allows changes of the input formula over time. Also, it can be considered as a prototypical Dynamic Constraint Satisfaction Problem (DCSP) [7].

Different methods have been applied to solve ISAT, among them, variations of the branch and bounds procedure, denoted as IDPL methods, which are usually based in the classical Davis-Putnam-Loveland (DPL) method. In a IDPL procedure, when adding new clauses, the procedure maintains the search tree generated previously for the set of clauses K. IDPL performs substantially faster than DPL for a large set of SAT problems [6]. Rather than solving related formulas separately, modern solvers attempt to solve them *incrementally* since many practical applications require solving a sequence of related SAT formulas [2,4].

Assuming an initial KB K, and a new CF ϕ to be added, let us consider some cases where SAT$(K \wedge \phi)$ can be determined efficiently.

1. If K and ϕ are 2-CF's then $(K \wedge \phi)$ is a 2-CF that is the input of ISAT. In this case, 2-ISAT is solvable in linear-time by applying the well known algorithms for 2-SAT [5,1]
2. For monotone formulas, ISAT keeps satisfiable formulas. If each variable maintains a unique sign in both K and ϕ then $(K \wedge \phi)$ is always satisfiable.
3. If ϕ consists of one clause and we have the searching graph of K, we only have to review which consistent path of the graph falsifies ϕ, and this can be done in linear time on the number of literals of K and the number of consistent paths of the searching graph.

It is clear that a set of changes over a satisfiable KB K in 2-CF could change K into a general CF, in which case, K will turn into a general CF K', $K \subset K'$, where the SAT problem on K' is a classic NP-complete problem.

From now on, let us consider that K is a 2-CF and ϕ is a 3-CF, both of them do not match the previous cases presented in this section. Therefore, we consider that ϕ consists of clauses that effectively decrease the set of models of K.

First, we show the relevance of our method to determine SAT$(K \wedge \phi)$ for applying it in a practical area. We consider automatic test pattern generation (ATPG) systems that consist in differentiating defective components from defect-free components. We start considering the method proposed by Larrabee [8]. Her method is based in the formation of cojunctive forms to express test patterns for single stuck at faults in combinatorial circuits.

For example, all binary and unary gates are expressed via CF's. Considering a logic gate with two inputs X, Y and output Z, basic gates are expressed as: *And* Gate: $(\overline{Z} + X)(\overline{Z} + Y)(Z + \overline{X} + \overline{Y})$, *Or* Gate: $(\overline{X} + Z)(\overline{Y} + Z)(X + Y + \overline{Z})$. The *Not* Gate: $(\overline{X}+Y)(\overline{Y}+X)$, and the *Xor* Gate is $(\overline{X}+Y+Z)(X+\overline{Y}+Z)(\overline{X}+\overline{Y}+\overline{Z})(X+Y+\overline{Z})$.

In Larrabee's method, a CF is used to generate test patterns on combinatorial circuits, considering the construction of such Boolean formula with true and false outputs of the circuit. In order to generate a test pattern for a single fault on the circuit, a CF that detects the fault in the circuit is extracted, and then, is needed to apply a procedure for reviewing the satisfiability of the formed formula.

The most important steps of the Larrabee's system are illustrated with the combinational circuit that appears in Figure 1:

1. A transformation process is applied on each gate forming the circuit and transforming it into a CF, according to the patterns described previously. For example, the first circuit in Figure 1 is equivalent to the CF: $\{\{C, E\}, \{\overline{C}, \overline{E}\}$ $, \{X, \overline{D}\}, \{X, \overline{E}\}, \{\overline{X}, D, E\}, \{\overline{D}, A\}, \{\overline{D}, B\}, \{D, \overline{A}, \overline{B}\}\}$.
2. It is needed to represent a faulted version of the initial circuit by making a copy of the original circuit, renaming variables and inserting two new nodes representing the presumed disrupted connection in the faulted circuit. In our example, we consider D' as a disrupted connection with a new output, denoted by X', on the final gate. In this case, D represents stuck-at 1, D' represents a faulted behavior at the fault site, and $\overline{D}$ is the node representing the correct behavior at the fault site. Then, we obtain the following CF: $\{\{C, E\}, \{\overline{C}, \overline{E}\}\, \{X', \overline{D}\}, \{X', \overline{E}\},$ $\{\overline{X'}, D, E\}, \{\overline{D}, A\}, \{\overline{D}, B\}, \{D, \overline{A}, \overline{B}\}, \{D'\}\}$.

3. The two circuits(faulted and unfaulted) are joined by a XOR-gate to represent that only one of them will be satisfiable. $F = \{\{X', \overline{D'}\}, \{X', \overline{E}\}, \{D'\}, \{\overline{X'}, D', E\}, \{C, E\}, \{\overline{C}, \overline{E}\}\ \{Z, Z'\}, \{\overline{Z}, \overline{Z'}\}, \{X, \overline{D}\}, \{X, \overline{E}\}\{\overline{D}, B\}, \{\overline{D}, A\}, \{\overline{X}, D, E\}, \{D, \overline{A}, \overline{B}\}, \{\overline{X}, X', Z\}, \{X, \overline{X'}, Z\}, \{\overline{X}, \overline{X'}, Z'\}, \{X, X', Z'\}\}$.
4. Our procedure requires that $\upsilon(\phi) \subseteq \upsilon(K)$, so we can redefine variables in ϕ that initially they do not appear in K, using binary clauses. In our example, $Z' = \overline{Z}$ and then the clauses $\{\overline{Z}, \overline{Z'}\}$ and $\{Z, Z'\}$ guarantee that only one of the two variables Z or Z' will be valid into the XOR gate: $\{\{\overline{X}, X', Z\}, \{X, \overline{X'}, Z\}, \{\overline{X}, \overline{X'}, Z'\}, \{X, X', Z'\}\}$.
5. At the end of the Larrabee's method, a final CF is obtained $F = \{\{X', \overline{D'}\}, \{X', \overline{E}\}, \{\overline{X'}, D', E\}, \{D'\}, \{C, E\}, \{\overline{C}, \overline{E}\}\ \{Z, Z'\}, \{\overline{Z}, \overline{Z'}\}, \{X, \overline{D}\}, \{X, \overline{E}\}, \{\overline{X}, D, E\}, \{\overline{D}, A\}, \{\overline{D}, B\}, \{D, \overline{A}, \overline{B}\}, \{\overline{X}, X', Z\}, \{X, \overline{X'}, Z\}, \{\overline{X}, \overline{X'}, Z'\}, \{X, X', Z'\}\}$

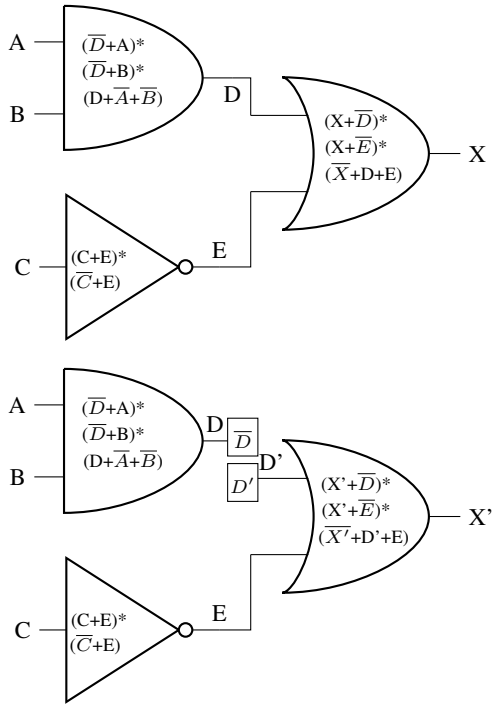

Fig. 1. Testing a combinatorial Circuit.

Now, we describe how our proposal works to review the satisifiability of $(K \wedge \phi)$, being K a 2-CF and ϕ a 3-CF. We illustrate our proposal for considering the final CF obtained via the Larrabee's method.

Let $S = S(K)$ be the base for the initial 2-CF K. First, we apply the simplification rules described in section 3, in order to reduce ϕ and extend K, keeping the logical

value of $(K \wedge \phi)$. After that, we consider $(\phi, s) = UP(\phi, S)$ because the partial assignment common to all model of K must remain in any model of ϕ. Furthermore, the new binary clauses in ϕ can be considered to be part of K and they can be deleted from ϕ.

Applying the rules to reduce formulas, we obtain the new two formulas:
$K = \{\{X', \overline{D'}\}, \{X', \overline{E}\}, \{D'\}, \{C, E\}, \{\overline{C}, \overline{E}\}\, \{Z, Z'\}, \{\overline{Z}, \overline{Z'}\}, \{X, \overline{D}\}, \{X, \overline{E}\},$
$\{\overline{D}, A\}, \{\overline{D}, B\}$ and $\phi = \{\{\overline{X'}, D', E\}, \{D, \overline{A}, \overline{B}\}, \{X, \overline{X'}, Z\}, \{\overline{X}, \overline{X'}, Z'\},$
$\{X, X', Z'\}\}$.

When $(\phi', S') = UP(\phi, S)$ is applied, new binary clauses are generated from clauses of ϕ. Those clauses are moved to K and deleted from ϕ. The process of moving binary clauses from ϕ to K, obligate us to update the closures and the base of K. Let us consider K' the set of new binary clauses to be added to K, such that $\upsilon(K') \subseteq \upsilon(K)$. $\forall c = \{x, y\} \in K'$, we consider the pair of implications: $\neg x \Rightarrow T(y)$ and $\neg y \Rightarrow T(x)$. Therefore, the original closures for x and y are updated as: $T(\neg y) = T(\neg y) \cup T(x)$ and $T(\neg x) = T(\neg x) \cup T(y)$.

Furthermore, $\forall T(l)$ where $\neg x \in T(l)$, it updates as $T(l) = T(l) \cup T(y)$, and $\forall T(l)$ where $\neg y \in T(l)$, it updates as $T(l) = T(l) \cup T(x)$. After updating the transitive closures, the new base for $K \cup K'$ has to be recomputed, and K is updated as: $K = K \cup K'$.

In the case of our example, we have that the base of our system of transitive closures is: $S(F) = \{D'\}$. By applying the rule of Pure Literal, D' is deleted from F, then
$F[D'] = \{\{X'\}, \{X', \overline{E}\}, \{C, E\}, \{\overline{C}, \overline{E}\}\, \{Z, Z'\}, \{\overline{Z}, \overline{Z'}\},$
$\{X, \overline{D}\}, \{X, \overline{E}\}, \{\overline{X}, D, E\}, \{\overline{D}, A\}, \{\overline{D}, B\}, \{D, \overline{A}, \overline{B}\}, \{\overline{X}, X', Z\}, \{X, \overline{X'}, Z\},$
$\{\overline{X}, \overline{X'}, Z'\}, \{X, X', Z'\}\}$.

As $\{X'\}$ is now a unitary clause, then X' is added to the base $S(F) = \{D', X'\}$, and $F[X']$ is computed. $F[X'] = \{\{C, E\}, \{\overline{C}, \overline{E}\}\, \{Z, Z'\}, \{\overline{Z}, \overline{Z'}\},$
$\{X, \overline{D}\}, \{X, \overline{E}\}, \{\overline{X}, D, E\}, \{\overline{D}, A\}, \{\overline{D}, B\}, \{D, \overline{A}, \overline{B}\}, \{X, Z\}, \{\overline{X}, Z'\}\}$.

The transitive closures are computed over the new K:
$T(A) = \{A\}, T(\overline{A}) = \{\overline{A}, \overline{D}\}\; T(B) = \{B\}, T(\overline{B}) = \{\overline{B}, \overline{D}\}$
$T(C) = \{C, \overline{E}\}, T(\overline{C}) = \{\overline{C}, E, X, Z', \overline{Z}\}$
$T(D) = \{D, A, B, X, Z', \overline{Z}\}, T(\overline{D}) = \{\overline{D}\}$
$T(E) = \{E, \overline{C}, X, Z', \overline{Z}\}, T(\overline{E}) = \{\overline{E}, C\}$
$T(X) = \{X, Z', \overline{Z}\}, T(\overline{X}) = \{\overline{X}, \overline{D}, \overline{E}, Z, C, \overline{Z'}\}$
$T(Z) = \{Z, \overline{Z'}, \overline{X}, \overline{E}, C, \overline{D}\}, T(\overline{Z}) = \{\overline{Z}, Z', X\}$
$T(Z') = \{Z', \overline{Z}, X\}, T(\overline{Z'}) = \{\overline{Z'}, Z, \overline{X}, \overline{E}, C, \overline{D}\}.$
And the last 3-CF is $\phi = \{\{\overline{X}, D, E\}, \{D, \overline{A}, \overline{B}\}\}$.

When $(\phi', S') = UP(\phi, S)$ is applied, then any variable $x \in \upsilon(S')$ does not appear more in ϕ', because if $x \in S'$ and $C = \{x, y, z\} \in \phi$ then C is satisfied and C does not appear more in ϕ'. Otherwise, if $C = \{\neg x, y, z\} \in \phi$ then the clause $\{y, z\}$ is generated instead of C, and it is added to K because this is a new binary clause. In whatever case, any variable in $S = Base(K \wedge \phi)$ does not appear more in ϕ'. Notice that during this step, a new base $S' \neq Base(K)$ could be generated since S' is looking for future feasible assignments for $(K \wedge \phi)$.

Our procedure looks afterwards for possible feasible assignments for $(K \cup \phi)$, in the next way: Let $S_1 = S \cup T(x)$ and $S_2 = S \cup T(\neg x)$ which are consistent because

in other case x or $\neg x$ must be in S.

Let $(F_4, S_4) = UP(\phi, S_1)$ and $(F_5, S_5) = UP(\phi, S_2)$. And let K_1, K_2 be the 2-CF's forming F_4 and F_5, respectively. Our algorithm consists of the following steps:

1. If $((Nil \in F_4$ or $(K \cup K_1)$ is unsatisfiable) and $(Nil \in F_5$ or $(K \cup K_2)$ is unsatisfiable)) then $(K \wedge \phi)$ is unsatisfiable. Because any feasible assignment can not be extended with value for the variable x without falsifying $(K \cup \phi)$.
2. Else If $(Nil \in F_4$ or $(K \cup K_1)$ is unsatisfiable) then $T(x)$ can not be part of any model of $(K \cup \phi)$, then we can extend the base S as: $S = S \cup T(\neg x)$.
3. Else If $(Nil \in F_5$ or $(K \cup K_2)$ is unsatisfiable) then $T(\neg x)$ can not be part of any model of $(K \cup \phi)$, then we can extend the base S as: $S = S \cup T(x)$.
4. Otherwise, all new unitary clause, generated via UP, allows to update the transitive closures. That is, $\forall\{l\}$ generated by $UP(\phi, S_1)$ or $UP(\phi, S_2)$, we have that $T(x) = T(x) \cup T(\neg l)$, and $T(l) = T(l) \cup T(\neg x)$. These four steps are iterated until determine the satisfiability of $(K \wedge \phi)$.

Applying these last steps to our example, we have that $UP(\phi, T(\overline{D})) = \phi[\overline{D}] = \{\{\overline{X}, E\}, \{\overline{A}, \overline{B}\}\}$ so that ϕ is reduced from a 3CF to a 2CF. Afterwards, the transitive closures have to be updated. This step finishes until K is unsatisfiable, or ϕ is empty (and then $K \cup \phi$ is satisfiable), or $UP(\phi, S)$ does not generate new binary or unitary clauses. Let us consider now that $UP(\phi, S)$ does not generate neither new unitary nor binary clauses, then as second step in our procedure, we select a literal $x \in ((Lit(K) \cap Lit(\phi)) - S)$ such that $T(x)$ and $T(\neg x)$ are consistent and $|T(x)| + |T(\neg x)|$ is maximum into the set of common literals of K and ϕ.

Again, considering our example, we have that the last closure system is: $T(A) = \{A, \overline{B}, \overline{D}\}$, $T(\overline{A}) = \{\overline{A}, \overline{D}\}$ $T(B) = \{B, \overline{A}, \overline{D}\}$, $T(\overline{B}) = \{\overline{B}, \overline{D}\}$
$T(C) = \{C, \overline{E}, \overline{X}, \overline{D}, Z, \overline{Z'}\}$, $T(\overline{C}) = \{\overline{C}, E, X, Z', \overline{Z}\}$
$T(D) = \{D, A, B, X, Z', \overline{Z}, \overline{B}, \overline{A}\}$ inconsistent
$T(\overline{D}) = \{\overline{D}\}$ thus $\overline{D}$ is added to S
$T(E) = \{E, \overline{C}, X, Z', \overline{Z}\}$, $T(\overline{E}) = \{\overline{E}, C, \overline{X}, \overline{D}, Z, \overline{Z'}\}$
$T(X) = \{X, Z', \overline{Z}, E, \overline{C}\}$, $T(\overline{X}) = \{\overline{X}, \overline{D}, \overline{E}, Z, C, \overline{Z'}\}$
$T(Z) = \{Z, \overline{Z'}, \overline{X}, \overline{E}, C, \overline{D}\}$, $T(\overline{Z}) = \{\overline{Z}, Z', X, E, \overline{C}\}$
$T(Z') = \{Z', \overline{Z}, X, E, \overline{C}\}$, $T(\overline{Z'}) = \{\overline{Z'}, Z, \overline{X}, \overline{E}, C, \overline{D}\}$

Note that in this case, the transitive closures obtained indicate that $\overline{D}$ is added to the base since an inconsistency was found for $T(D)$. Therefore, K has various models that satisfy ϕ. But, if we consider D as part of a model of K, all ternary clause contained in ϕ is transformed into a binary clause, and the process is finished indicating that $(K \wedge \phi)$ is satisfiable.

The great advantage of this method is that at least two thirds of the clauses generated are binary clauses (the 2-CF). This is true because each two-input unate gate contributes two binary clauses and one ternary clause. Unate gates with more than two inputs contribute more than two thirds of binary clauses. Fanout points, buffers, and inverters contribute with only binary clauses. In practice, applying this method, at most 80% to 90% of the clauses are binary clauses. Thus, our algorithm provides an efficient method to solve this class of problems.

6 Conclusions

We have designed a novel method for the incremental satisfiability (ISAT) problem. Thus, we have shown different cases where the ISAT problem can be solved in polynomial time.

Especially, considering an initial base K in 2-CF, we present an algorithm for solving the 2-ISAT problem that allows us to determine the satisifiability for $(K \wedge \phi)$, where ϕ is a 3-CF. Furthermore, we have established some tractable cases for the 2-ISAT problem.

We have illustrated the usefulness of our method in the area of automatic test pattern generation (ATPG) systems that allows to distinguish defective components from defect-free components in combinatorial circuits.

References

1. Buresh-Oppenheim, J., Mitchell, D.: Minimum 2CNF resolution refutations in polynomial time. Proc. SAT'07 – 10th int. Conf. on Theory and applications of satisfiability testing. 300–313 (2007)
2. Cabodi, G., Lavagno, L., Murciano, M., Kondratyev, A., Watanabe, Y.: Speeding-up heuristic allocation, scheduling and binding with SAT-based abstraction/refinement techniques. ACM Trans. Design Autom. Electr. Syst. 15(2) (2010)
3. De Ita, G., Marcial-Romero, R., Hernández, J. A.: The Incremental Satisfiability Problem for a Two Conjunctive Normal Form, *Ceur-WS Lanmr 2016*. 1659, 25–32 (2016)
4. Eén, N., Sorensson, K.: An Extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, Selected Revised Papers of 6th International Conference on Theory and Applicationsof Satisfiability Testing (SAT), Santa Margherita Ligure, Italy, LNCS. 2919, 502–518 (2003)
5. Gusfield, D., Pitt, L.: A Bounded Approximation for the Minimum Cost 2-Sat Problem. Algorithmica 8. 103–117 (1992)
6. Hooker, J.N.: Solving the incremental satisfiability problem. Journal of Logic Programming. 15, 177–186 (1993)
7. Gutierrez, J., Mali, A.: Local search for incremental Satisfiability. Proc. Int. Conf. on AI (IC–AI'02), Las Vegas. 986–991 (2002)
8. Larrabee, T.: Test Pattern Generation Using Boolean Satisfiability. IEEE Trans. on Computer–Aided Design of Integrated Circuits and Systems. 11(1), 4–15 (1992)
9. Mouhoub, M., Sadaoui, S.: Systematic versus non systematic methods for solving incremental satisifiability, Int. J. on Artificial Inteligence Tools. 16(1), 543–551 (2007)
10. Wieringa, S.: Incremental Satisfiability Solving and its Applications. PhD thesis, Department of Computer Science and Engineering, Alto University Pub. (2014)