On Bridging Simulation and Formal Verification

Eugene Goldberg

Cadence Research Labs, USA, 2150 Shattuck Ave., 10^{th} floor, Berkeley, California,
94704, phone: (510)-647-2825, fax: (510)-647-2801 `egold@cadence.com`

Abstract. Simulation and formal verification are two complementary techniques for checking the correctness of hardware and software designs. Formal verification proves that a design property holds for all points of the search space while simulation checks this property by probing the search space at a subset of points. A known fact is that simulation works surprisingly well taking into account the negligible part of the search space covered by test points. We explore this phenomenon by the example of the satisfiability problem (SAT). We believe that the success of simulation can be understood if one interprets a set of test points not as a sample of the search space, but as an "encoding" of a formal proof[1] We introduce the notion of a *sufficient* test set of a CNF formula as a test set encoding a formal proof that this formula is unsatisfiable. We show how sufficient test sets can be built. We discuss applications of *tight* sufficient test sets for testing technological faults (manufacturing testing) and design changes (functional verification) and give some experimental results.

1 Introduction

Development of new methods of hardware and software verification is in growing demand due to ever-increasing design complexity. Simulation and formal verification are two complementary verification techniques. Given a design property ξ, formal verification proves that ξ holds for every point of the search space. Simulation verifies ξ by testing a small subset of the search space. The main drawback of formal verification is its unscalability while an obvious flaw of simulation is its inability to prove that ξ holds for every point of the search space. Nevertheless, the main bulk of verification is currently done by simulation: it is scalable and works surprisingly well even though the set of test points (further referred to as the **test set**) comprises a negligible part of the search space.

We study why simulation is so effective on the example of the satisfiability problem (SAT). In terms of SAT, formal verification is to prove that a CNF formula $F(x_1,.., x_n)$ is unsatisfiable at every point $\boldsymbol{p} \in \{0,1\}^n$. On the other hand, simulation is to give some guarantee that F is unsatisfiable by testing it at a (small) set of points from $\{0,1\}^n$. (Local search algorithms pioneered in [5, 6] can be viewed as solving SAT by "simulation". While these algorithms target *satisfiable* formulas, in this paper, we are mostly interested in applying

---

[1] In VMCAI-08 proceedings we mistakenly used the term "encryption".

simulation to *unsatisfiable* formulas.) We believe that the success of simulation can be explained if one interprets a test set not as a sample of the search space but as an *"encoding"* of a formal proof that the CNF formula under test is unsatisfiable.

We introduce procedure $Sat(T,F,L)$ that checks satisfiability of a CNF formula F using a test set T and a set L of lemma clauses (or just lemmas for short). Henceforth we will also refer to a set of lemma clauses L as a **proof**. $Sat(T,F,L)$ is not *a practical procedure* and is introduced just to formally define what it means that a test set T encodes a proof L . Namely, $\boldsymbol{T}$ **encodes** $\boldsymbol{L}$ if $Sat(T,F,L)$ proves F to be unsatisfiable.

The set of lemma clauses $L_1,\dots,L_k$ is ordered and the last clause L_k is empty. The $Sat(T,F,L)$ procedure is based on the fact that a CNF formula is unsatisfiable iff it has a stable set of points (SSP) [4]. In this paper, we introduce an efficient procedure that, given a CNF formula F' and a set of points T, checks if T contains an SSP of F'. This procedure is used by $Sat(T,F,L)$ to prove that F implies L_i. This is done by checking if the set T contains an SSP for a CNF formula F' equivalent to $F \rightarrow L_i$. If $F \rightarrow L_i$ holds, clause L_i is added to F. Both L and T are crucial for $Sat(T,F,L)$. The set L specifies a "high-level structure" of the proof by indicating the set of lemmas to prove. On the other hand, the set T is necessary for proving the lemmas of L efficiently.

A test set T is called **sufficient** for a CNF formula F, if there is a set of lemma clauses L for which $Sat(T,F,L)$ proves unsatisfiability of F. The fewer lemmas a sufficient test set T needs for proving unsatisfiability of F by $Sat(T,F,L)$, the larger the size and the higher the quality of T is. If the set L of lemma clauses consists only of an empty clause, $Sat(T,F,L)$ succeeds in proving unsatisfiability of F only if T contains an SSP. So an SSP is a test set of the highest quality but it usually contains an exponential number of points [4]. In [3], we introduced the notion of a *point image* of resolution proof R that a CNF formula is unsatisfiable. We show in this paper that if the clauses of L are the resolvents of R, the procedure $Sat(T,F,L)$ succeeds if T is a point image of R. A point image of a resolution proof is a sufficient test set of lower quality but it contains dramatically fewer points than an SSP.

A sufficient test set may occupy a negligible part of the search space. (For example, a point image of a resolution proof is at most two times the size of the proof.) This fact sheds light on why simulation works so well even though it samples only a tiny portion of the search space. A cleverly selected set of tests (e.g. tests exercising various corner cases) may specify a set of points that encode a formal proof that the property in question holds (or a "significant part" of such a proof).

Simulation can be used for two kinds of problems. We will refer to the problems of the first kind as *property checking*. In the context of SAT, property checking by simulation is to prove the satisfiability of a CNF formula F by probing the value of F at a (small) set of points or to give some "guarantee" that F is unsatisfiable. The problems of the second kind are referred to as *property preservation*. In the context of SAT, property preservation is as follows.

Suppose that F is an unsatisfiable formula and we need to find a (small) set of points T such that a satisfiable formula F' obtained by a (small) variation of F most likely evaluates to 1 for a point $\boldsymbol{p}$ of T. In other words, we want to find a set of points T that will most likely identify satisfiable variations of F. Assuming that F describes a design property, the variation of F may specify a design change (if we deal with a software model of the design) or a technological fault (if F describes a hardware implementation of the design).

Although the theory we develop can be applied to the problems of both kinds, the main focus of this paper is property preservation. (Some insights into how sufficient test sets can be used for property checking are given in [2].) The main idea is as follows. Let R be a proof that F is unsatisfiable. To build a test set that detects satisfiable variations of F we propose to extract tests from a tight sufficient test set T specified by R. Informally, a sufficient test set is **tight** if points of T falsify as few clauses of F as possible. (Since F is unsatisfiable, obviously a point of T has falsify at least one clause of F). "Regular" tests i.e. input assignments are extracted from the points of T. (If F describes a property of a circuit N, then a point $\boldsymbol{p}$ of T is a complete an assignment to the variables of N. By dropping all the assignments of $\boldsymbol{p}$ but the input assignments of N we obtain a regular test vector. In more detail, the relation between regular tests and points is described in Section 5). If R is a resolution proof, then tests are extracted from a tight point image of R.

As a practical application of our theory we study regular tests (i.e. input assignments) extracted from a tight point image T of a resolution proof that two copies of a circuit N are functionally equivalent. We show that such regular tests detect the testable stuck-at faults of N. This result explains why the stuck-at fault model is so successful. Besides, this result suggests that the success of this model may have nothing to do with the assumption made by many practitioners that the stuck-at fault model works so well because it correctly describes the "real" faults. Interestingly, tests extracted from T may detect the same stuck-at fault many times (i.e. for the same stuck-at fault different test vectors may be generated). At the same time, in [8] it was shown experimentally, that test sets where the same stuck-at fault was tested many times had the best performance in identifying faulty chips.

In the experimental part of this paper, we apply tests extracted from a resolution proof that two copies of a circuit are identical to detection of literal appearance faults (such faults are more subtle than stuck-at faults). Our results show that tests extracted from resolution proofs have much higher quality than random tests.

This paper is structured as follows. Section 2 describes a procedure for checking if a set of points contains an SSP of a CNF formula. In Section 3, we describe the procedure $Sat(T,F,L)$ and introduce the notion of a sufficient test set. Generation of tight sufficient test sets is described in Section 4. In Section 5, we discuss the specifics of testing formulas describing circuits. Section 6 describes application of sufficient test sets for testing design changes and manufacturing faults. We give some experimental results in Section 7 and conclude by Section 8.

2 Checking if Test Set Contains SSP

In this section, we give some basic definitions, recall the notion of a stable set of points (SSP) [4] and introduce a procedure that checks if a set of points contains a stable subset.

2.1 Basic Definitions

Let F be a CNF formula (i.e. conjunction of disjunctions of literals) over a set X of Boolean variables. The satisfiability problem (SAT) is to find a complete assignment $\boldsymbol{p}$ (called a **satisfying assignment**) to the variables of X such that $F(\boldsymbol{p}) = 1$ or to prove that such an assignment does not exist. If F has a satisfying assignment, F is called **satisfiable**. Otherwise, F is **unsatisfiable**. A disjunction of literals is further referred to as a **clause**. A complete assignment to variables of X will be also called a **point** of the Boolean space $\{0,1\}^{|X|}$. A point $\boldsymbol{p}$ **satisfies** clause C, if $C(\boldsymbol{p})$=1. If $C(\boldsymbol{p})$=0, $\boldsymbol{p}$ is said to **falsify** C. Denote by $\mathit{Vars}(C)$ and $\mathit{Vars}(F)$ the set of variables of C and F, respectively. We will call a complete assignment $\boldsymbol{p} \in \{0,1\}^{|X|}$ **a test** for F. We will call a set of points $T \subseteq \{0,1\}^{|X|}$ **a test set** for F.

2.2 Stable Set of Points

Let a point $\boldsymbol{p} \in \{0,1\}^{|X|}$ falsify a clause C of k literals. Denote by $\mathit{Nbhd}(\boldsymbol{p},C)$ the set of k points obtained from $\boldsymbol{p}$ by flipping the value of one of k variables of C. For example, let X=$\{x_1,.., x_5\}$ and $C = x_2 \vee x_3 \vee \overline{x_5}$ and $\boldsymbol{p}$=(x_1=0, x_2=0, x_3=0, x_4=1, x_5=1). (Note that $C(\boldsymbol{p})$=0.) Then $\mathit{Nbhd}(\boldsymbol{p},C)$ =$\{\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3\}$ where $\boldsymbol{p}_1$ = (.., x_2=1,..), $\boldsymbol{p}_2$=(.., x_3=1,..), $\boldsymbol{p}_3 = (\dots, x_5$=0). (For each $\boldsymbol{p}_i$, the skipped assignments are the same as in $\boldsymbol{p}$.)

Let a CNF formula F over a set X of Boolean variables consist of clauses $C_1,\dots,C_s$. Let $T = \{\boldsymbol{p}_1,\dots,\boldsymbol{p}_m\}$ be a non-empty set of points from $\{0,1\}^{|X|}$ such that $F(\boldsymbol{p}_i)$=0, i=1,..,m. The set T is called a **stable set of points** (**SSP**) of F if for each $\boldsymbol{p}_i \in T$, there is a clause C_k of F such that $C_k(\boldsymbol{p}_i)$=0 and $\mathit{Nbhd}(\boldsymbol{p}_i,C_k) \subseteq T$. (In [4] we used a slightly different but equivalent definition of SSP.)

Proposition 1. *Let* F=$\{C_1,..,C_s\}$ *be a CNF formula over a set* X *of Boolean variables. Formula* F *is unsatisfiable iff there is a set* T *of points from* $\{0,1\}^{|X|}$ *such that* T *is an SSP of* F.

Proof is given in [4].

2.3 Checking if a test set contains an SSP

Given a set of points T and a CNF formula F, checking if T is an SSP for F is very simple. One just needs to check if for every point $\boldsymbol{p}$ of T there is a clause C of F such that $\mathit{Nbhd}(\boldsymbol{p},C) \subseteq T$. If the check passes, then T is an SSP for F

and hence the latter is unsatisfiable. The complexity of this check is $|T|*|F|*|X|$ where X is the set of variables of F.

It is quite possible that a subset of T is an SSP of F while T itself is not. The procedure of **Figure 1** checks if there is a subset of T that is an SSP of F. For every point $\boldsymbol{p}$ of T it checks if there is a clause C of F such that $Nbhd(\boldsymbol{p},C) \subseteq T$ (the function $no\_clause(\boldsymbol{p},F,T)$). If such a clause does not exist, $\boldsymbol{p}$ is removed from T and every point of T is checked again. (The reason for starting over again is as follows. Suppose that in the previous iterations a point $\boldsymbol{p}^*$ was not removed from T because for some clause C of F, $Nbhd(\boldsymbol{p}^*,C) \subseteq T$. If $\boldsymbol{p}$ was in $Nbhd(\boldsymbol{p}^*,C)$), then removing $\boldsymbol{p}$ from T would break the relation $Nbhd(\boldsymbol{p}^*,C) \subseteq T$.)

```
Stable_subset_check($T$,$F$)
  {removed=true;
  while (removed)
    {removed=false;
    for (every point $\boldsymbol{p} \in T$)
      if (no_clause($\boldsymbol{p}$,$F$,$T$))
        {$T = T \setminus \{\boldsymbol{p}\}$;
        removed=true;
        break;}}
  if ($T \neq \emptyset$) return (stable)
  else return(unstable);}
```

Figure 1. Checking if T contains an SSP

This repeats until no point is removed from T, which may happen only in two cases: a) T is empty (and so the original set T did not contain a stable subset); b) The remaining points of T form an SSP. The complexity of this procedure is $|T|^2*|F|*|X|$.

3 Procedure *Sat*(*T*,*F*,*L*) and Sufficient Test Sets

In this section, we describe a procedure $Sat(T,F,L)$ that uses a test set T to prove that a CNF formula F is unsatisfiable. $Sat(T,F,L)$ is *not a practical procedure.* We introduce it just to formally define what it means that T encodes a proof L. We also introduce the notion of a sufficient test set and describe how sufficient test sets can be obtained.

3.1 *Sat*(*T*,*F*,*L*) procedure

The pseudocode of the procedure $Sat(T,F,L)$ is shown in **Figure 2**. Here L is a set of lemma clauses L_1,.., L_k where the clause L_k is empty. First, *Sat(T,F,L)* checks if a point $\boldsymbol{p}$ of T satisfies F. If such a point exists, $Sat(T,F,L)$ reports that F is satisfiable. Then $Sat(T,F,L)$ processes the clauses of L in the order they are numbered. For every lemma clause L_i of L, this procedure checks if F implies L_i, by calling the function $implies(T,F,L_i)$. If it succeeds in proving this implication, L_i is *added* to F. To check if F implies L_i, the function *implies*(T,F,L_i) uses the procedure *Stable_subset_check* of **Figure 1** as follows. First, the subformula F_{L_i} is obtained from F by making the assignments setting all the literals of L_i to 0. Formula F implies L_i iff F_{L_i} is unsatisfiable. To check if F_{L_i} is unsatisfiable, the procedure *Stable_subset_check*(T_{L_i},F_{L_i}) is called by the function *implies*(T,F,L_i) where T_{L_i} is the subset of points of T falsifying L_i. This procedure checks if the set T_{L_i} contains a subset that is an SSP with respect to F_{L_i}. The complexity of *Sat*(T,F,L) is $|T|^2 * |F| * |X| * |L|$ where X is the set of variables of F and $|L|$ is the number of lemma clauses. (In [2], we give a version of *Sat*(T,F,L) that is linear in $|T|$ but needs more information than the procedure of **Figure 2**.)

```
Sat(T,F,L)
  {if (satisfy(T,F)) return(sat)
  for (i=1,..,k))
    {if (implies(T,F,L_i)==false)
      return(unknown)
    F = F ∪ {L_i} }}
  return(unsat);}
```

Figure 2. Pseudocode of procedure *SAT*(T,F,L)

3.2 Sufficient test sets

We will say that a test set T is **sufficient** for F, if there is a set L of lemma clauses such that *Sat*(T,F,L) succeeds in proving the unsatisfiability of F. That is, T is a sufficient test set for F, if it has enough points to show that F is unsatisfiable by proving a sequence of lemmas L.

In general, the fewer lemma clauses are in the set L, the larger test set T is necessary for *Sat*(T,F,L) to succeed. In particular, if L contains only an empty clause, then *Sat*(T,F,L) succeeds only if T contains an SSP. On the other hand, as we show below, if L consists of the resolvents of a resolution proof R that F is unsatisfiable, $Sat(T, F, L)$ succeeds even if T is just a point image of R.

A resolution proof is an ordered set of resolution operations that proves unsatisfiability of a CNF formula F by deriving an empty clause [9]. **A resolution operation** is performed over two clauses C' and C'' such that a) they have opposite literals of some variable x_i and b) there is only one such variable for C' and C''. The result of the resolution operation is a clause C called the **resolvent** of C' and C''. The resolvent C consists of all the literals of C' and C'' but the literals of x_i. (C is said to be obtained by resolving C' and C'' in variable x_i.) For example, if C'=$x_2 \vee \overline{x_4} \vee x_{20}$ and C''=$x_4 \vee \overline{x_{31}}$, then by resolving them in variable x_4 we obtain the resolvent C= $x_2 \vee x_{20} \vee \overline{x_{31}}$.

The notion of a point image of a resolution proof R was introduced in [3]. A set of points T is called a **point image of *R*** if for any resolution operation of R over clauses C' and C'', there are points $\boldsymbol{p'}$,$\boldsymbol{p''} \in T$ satisfying the following two conditions: a) $C'(\boldsymbol{p'})= C''(\boldsymbol{p''})$=0; b) $\boldsymbol{p'}$,$\boldsymbol{p''}$ are different only in the variable in which clauses C' and C'' are resolved. Such two points are called **a point image of the resolution operation** over C' and C''.

Now we show that if R is a resolution proof that F is unsatisfiable and T is a point image of R, then $Sat(T, F, L)$ returns *unsat* where L is the set of resolvents of R. Let C be a resolvent of R obtained by resolving C' and C''. Then C is in L. When the $Sat(T,F,L)$ procedure gets to proving that C is implied by the current formula F, clauses C' and C'' are in F. Let F_C be the formula obtained from F (by making the assignments setting the literals of C to 0) for checking if F implies C. In F_C, clauses C' and C'' turn into unit clauses x_i and $\overline{x_i}$ (where x_i is the variable in which C' and C'' are resolved). Then the points $\boldsymbol{p'}$,$\boldsymbol{p''}$ form an SSP with respect to these unit clauses and hence with respect to F_C. So the procedure $Sat(T,F,L)$ succeeds in proving unsatisfiability of F. A point image is a weak sufficient test set, because it can be used only to prove very simple lemmas (that the resolvent of C' and C'' is implied by $C' \wedge C''$).

3.3 Generation of sufficient test sets

Given a CNF formula F, one can build its sufficient test set as a point image T of a resolution proof R that F is unsatisfiable. Building T is very simple. For every pair of clauses C' and C'' whose resolvent is in R one just needs to find a point image of the resolution operation over C' and C''. The union of point images of all resolution operations forms a point image of R (and so a sufficient test set for F). The size of such a point image is twice the size of R at most.

As we mentioned above, a point image of a resolution proof R is a weak sufficient test set. However, one can always get a stronger test set by "rarefying" R. The idea is to remove some resolvents from R (preserving an empty clause) and use the remaining clauses as the set L of lemmas. Then for every clause L_i of L we build an SSP S_i for F_{L_i} thus proving that $F \rightarrow L_i$. (We assume that the lemma clauses L_1,.., L_{i-1} proved before L_i have been added to F.) A procedure for building an SSP is described in [4]. Since some resolvents of R are missing, now one may need more than two points to prove that $F \rightarrow L_i$. The set $T = S_1 \cup \, .. \, \cup S_k$ where $k = |L|$ forms a sufficient test set that is stronger than a point image of R (because T can prove more complex lemmas). If one removes from R all the resolvents but an empty clause, T turns into an SSP.

4 Tight Sufficient Test Sets

The fact that a test set T is sufficient for a CNF formula F means that T is complete in the sense that it encodes a proof that F is unsatisfiable. However, this completeness alone does not make T a high-quality test set for a property preservation problem. Recall that we are interested in finding a test set such that, given an unsatisfiable formula F, it will most likely "detect" satisfiable variations of F. In other words, given a satisfiable formula F' obtained from F by a small change, we want T to contain a point $\boldsymbol{p}$ that satisfies F' and so detects this change. This can be done by making sufficient test sets tight. Informally, a sufficient test set T is **tight** if every point $\boldsymbol{p}$ of T falsifies as few clauses of the *original* formula F as possible. (Ideally, every point $\boldsymbol{p}$ of T should falsify only one original clause). The intuition here is that if $\boldsymbol{p}$ falsifies only clause C_i of F, then $\boldsymbol{p}$ may detect a variation of F that includes disappearance of C_i from F (or adding to C_i a literal satisfied by $\boldsymbol{p}$).

Let us consider building a tight point image T of a resolution proof R. Let C be the resolvent of C' and C''. When looking for two points $\boldsymbol{p'}$,$\boldsymbol{p''}$ forming a point image of the resolution operation over clauses C' and C'' (and so forming an SSP of sub formula F_C) we have freedom in assigning variables of F that are not in C' and C''. To make the test set T tight, these assignments should be chosen to minimize the number of clauses falsified by $\boldsymbol{p'}$,$\boldsymbol{p''}$. Note that since $\boldsymbol{p'}$,$\boldsymbol{p''}$ are different only in one variable (in which C' and C'' are resolved), picking one point, say $\boldsymbol{p'}$, completely determines the point $\boldsymbol{p''}$. This poses the following problem. It is possible that no matter how well one picks the point $\boldsymbol{p'}$ to falsify only one clause of F, the corresponding point $\boldsymbol{p''}$ falsifies many clauses of F.

In [2], we describe a solution to the problem above. Namely we describe a version of the procedure *Sat*(T,F,L) that slightly "relaxes" the definition of a sufficient test set. (By changing procedure *Sat*(T,F,L), we essentially change the definition of proof encoding we use. Obviously, the same proof can be encoded in many ways.) In this version, in points $\boldsymbol{p'}$,$\boldsymbol{p''}$, only the parts consisting of the assignments of the variables of $\mathit{Vars}(C') \cup \mathit{Vars}(C'')$ have to be at Hamming distance 1 (i.e. one just needs to guarantee that both $\boldsymbol{p'}$,$\boldsymbol{p''}$ falsify the resolvent of C' and C''). Assignments to the variables that are not in C' and C'' can be done *independently* in $\boldsymbol{p'}$,$\boldsymbol{p''}$. (In [2], we also describe how to extract a tight sufficient test set from a "rarefied" resolution proof introduced in subsection 3.3, i.e. how to build tight sufficient tests sets that are *stronger* than those obtained from resolution proofs.)

5 Circuit Testing

So far we have studied the testing of *general* CNF formulas. In this section, we consider the subproblem of SAT called Circuit-SAT. In this subproblem, CNF formulas describe *combinational circuits.* In this section, we discuss some specifics of testing formulas of Circuit-SAT.

5.1 Circuit-SAT

Let N be a single-output combinational circuit. Let F_N be a CNF formula specifying N and obtained from it in a regular way. That is for every gate G_i,i=1,..,k of the circuit N, a CNF formula $F(G_i)$ specifying G_i is formed and $F_N = F(G_1) \wedge \ldots \wedge F(G_k)$. For example, if G_i is an AND gate implementing $v_i = v_m \wedge v_n$ (where v_i, v_m,v_n describe the output and inputs of G_i), $F(G_i)$ is equal to $(\overline{v_m} \vee \overline{v_n} \vee v_i) \wedge$ $(v_m \vee \overline{v_i}) \wedge$ $(v_n \vee \overline{v_i})$. Let variable z describe the output of N. Then the formula $F_N \wedge z$ (where z is just a single-literal clause) is satisfiable iff there is an assignment to input variables of N for which the latter evaluates to 1. We will refer to testing the satisfiability of F_N $\wedge z$ as **Circuit-SAT**.

5.2 Specifics of testing Circuit-SAT formulas

Let $N(Y,H,z)$ be a circuit where Y, H are the set of input and internal variables respectively. Let $F_N \wedge z$ be a CNF formula describing the instance of *Circuit-SAT* specified by $N(Y,H,z)$. Let $\boldsymbol{p}$ be a test as we defined it for SAT (i.e. a complete assignment to the variables of $Y \cup H \cup \{z\}$. We will denote by $inp(\boldsymbol{p})$ **the input part** of $\boldsymbol{p}$ that is the part consisting of the assignments of $\boldsymbol{p}$ to the variables of Y.

The main difference between the definition of a test as a complete assignment $\boldsymbol{p}$ that we used so far and the one used in circuit testing is that in circuit testing *the input part* of $\boldsymbol{p}$ is called a test. (We will refer to $inp(\boldsymbol{p})$ as a **circuit test**.) The reason for that is as follows. Let $N(Y,H,z)$ be a circuit and $F_N \wedge z$ be the CNF formula to be tested for satisfiability. A complete assignment $\boldsymbol{p}$ can be represented as $(\boldsymbol{y},\boldsymbol{h},z^*)$ where $\boldsymbol{y}$, $\boldsymbol{h}$ are complete assignments to Y, H respectively and z^* is an assignment to variable z. Denote by F the formula $F_N \wedge z$. If $F(\boldsymbol{p})$=0, then no matter how one changes assignments $\boldsymbol{h}$, z^* in $\boldsymbol{p}$, the latter falsifies a clause of F. (So, in reality, $inp(\boldsymbol{p})$ is a cube specifying a huge number of complete assignments.) Then instead of enumerating the complete assignments to *Vars*(F) one can enumerate the complete assignments to the set Y of input variables. In our approach, however, using cubes is unacceptable because the complexity of *Sat*(T,F,L) is proportional to the size of T.

Note that, given a sufficient test set $T=\{\boldsymbol{p_1},\dots,\boldsymbol{p_m}\}$, one can always form a circuit test set $inp(T)=\{\boldsymbol{y_1},\dots,\boldsymbol{y_k}\}$, $k \leq m$, consisting of input parts of the points from T. (Some points of T may have identical input parts, so $inp(T)$ may be smaller than T.) In the case of manufacturing testing, transformation of T into $inp(T)$ is mandatory. In this case, a *hardware* implementation of a circuit N is tested and usually one has access only to the input variables of N. (In the case of functional verification, one deals with a *software model* of N and so any variable of F can be assigned an arbitrary value.)

A point $\boldsymbol{p}_i$ of T has an interesting *interpretation* in Circuit-SAT if the value of z is equal to 1 in $\boldsymbol{p}_i$. Let F' be the subset of clauses of F_N falsified by $\boldsymbol{p_i}$. (For a tight test set, F' consists of a very small number of clauses, most likely one clause.) Suppose N has changed (or has a fault) and this change can be simulated by removing the clauses of F' from F_N or by adding to every clause of F' a literal satisfied by $\boldsymbol{p}_i$. Then $\boldsymbol{p_i}$ satisfies the modified formula F. So the internal part of $\boldsymbol{p}_i$ specifies the change that needs to be brought into circuit N to make $inp(\boldsymbol{p}_i)$ a circuit test that detects the satisfiability of the modified N.

6 Testing design changes/manufacturing faults

In this section, we consider the problem of property preservation (i.e. the problem of testing design changes and manufacturing faults) in more detail. In terms of SAT, the objective of property preservation is to detect a satisfiable variation (fault) of an unsatisfiable CNF formula F. We assume here that F specifies a property ξ of a circuit N. The idea of our approach is to build a resolution proof

R that F is unsatisfiable and then use R (possibly "rarefied") to build a *tight* sufficient test set T. This test set is meant to detect changes/faults that break the property ξ. Every point $\boldsymbol{p}_i$ of T can be trivially transformed to a circuit test by taking the input part of $\boldsymbol{p}_i$. For the sake of clarity, in the following write-up we consider the testing of manufacturing faults (however the same approach can be used for verifying design changes).

Usually, to make manufacturing test generation more efficient, a fault model (e.g. the stuck-at fault model [1]) is considered. Then a set of tests detecting all testable faults of this model is generated. An obvious flaw of this approach is that one has to foresee what kind of faults may occur in the circuit. Nevertheless, some fault models (especially the stuck-at fault model) are widely used in industry. The reason for such popularity is that a set of tests detecting all testable stuck-at faults also detects a great deal of faults of other types. An obvious advantage of our approach is that it is *fault model independent*. So one does not need to guess what may happen with the chip.

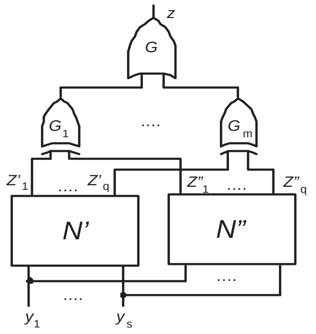

Fig. 3. Miter M of circuits N' and N''

For the case of generality, we consider the situation when one does not know any specific property of the circuit N to be tested. In this case, one can employ the most fundamental property of a circuit which is its self-equivalence. In this section, we show that a tight sufficient test set T for the formula specifying self-equivalence of N contains tests for detecting stuck-at faults. (In [2], we prove that on the one hand, *inp*(T) contains tests for detecting *all testable stuck-at faults*, on the other hand, *inp*(T) is *stronger* than a set of tests detecting all testable stuck-at faults.) These results offer a good explanation of why test sets detecting stuck-at faults work so well for other types of faults.

Further exposition is structured as follows. First we describe a circuit (called a miter) that is used for equivalence checking. Then we give the definition of a stuck-at fault in circuit N. After that we show how one can build a test detecting a stuck-at fault using a formula F that describes checking self-equivalence of N. Finally, we show that a tight point image of a "natural" resolution proof that F is unsatisfiable contains such tests.

6.1 Manufacturing tests and self-equivalence check

Fig. 3 shows a circuit M (called **a miter**) composed of two s-input, q-output circuits N' and N''. Here G_i is an XOR gate and G is an OR gate. The circuit M evaluates to 1 iff N' and N'' produce different output assignments for the same input assignment. So N' and N'' are functionally equivalent iff the CNF formula $F_M \wedge z$ is unsatisfiable (here F_M specifies the functionality of M and z is the output variable of M).

Suppose that we want to generate a set of manufacturing tests for a circuit N. We can do this as follows. First we build the miter M of two copies of N. (In this case, N' and N'' of Fig. 3 are just copies of N having the same input variables and separate sets of internal variables.) After that we construct a proof R that the formula $F = F_M \wedge z$ is unsatisfiable and then use R to build a *tight* sufficient test set T. The idea is that being tight, T can be used for detection of variations of F describing appearance of a fault in one of the copies of N.

6.2 Stuck-at faults

A stuck-at fault in a circuit N, describes the situation when a line in N is stuck at *constant* value 0 or 1. Let $G_i(v_m,v_k)$ be a gate of N. Then appearance of a stuck-at-1 fault ϕ on the input line v_m of G_i, means that for every assignment to the inputs of N the value of v_m remains 1. (Suppose that the output of gate G_m described by variable v_m, in addition to an input of G_i, feeds an input of some other gate G_p. In the *single* stuck-at fault model we use in this paper, only the input v_m of G_i or G_p is assumed to be stuck at a constant value. However, if the *output* line of G_m is stuck at 1, then input lines v_m of both G_i and G_p are stuck at 1.) Let G_i be an AND gate. Then the functionality of G_i can be described by CNF $F(G_i) = (\overline{v_m} \vee \overline{v_k} \vee v_i) \wedge (v_m \vee \overline{v_i}) \wedge (v_k \vee \overline{v_i})$ where v_i describes the output of G_i. The fault ϕ above can be simulated by removing the clause $v_m \vee \overline{v_i}$ from $F(G_i)$ (it is satisfied by v_m=1) and removing the literal $\overline{v_m}$ from the clause $\overline{v_m} \vee \overline{v_k} \vee v_i$ of $F(G_i)$.

6.3 Construction of tests detecting stuck-at faults

Suppose the stuck-at-1 fault ϕ above occurred in the copy N' of N (i.e. it occurred on the input line v'_m of the AND gate $G_i(v'_m,v'_k)$ of N'). Let us show how this fault can be detected using the formula F=$F_M \wedge z$. Let $\boldsymbol{p}$ be an assignment falsifying the clause $v'_m \vee \overline{v'_i}$ of $F(G'_i)$ and satisfying *every other* clause of F. Then the input assignment *inp*($\boldsymbol{p}$) is a circuit test detecting ϕ. Indeed, since $\boldsymbol{p}$ satisfies all the clauses of F but $v'_m \vee \overline{v'_i}$, then N'' (the correct copy of N) and N' (the faulty copy) produce different output assignments. Besides, since $\boldsymbol{p}$ falsifies $v'_m \vee \overline{v'_i}$ and satisfies the clause $v'_k \vee \overline{v'_i}$ the assignments to the variables of G'_i are v'_m=0,v'_k=1, v'_i=1. That is the output of G'_i has exactly the value, that would have been produced if v'_m got stuck at 1. If there is no point $\boldsymbol{p}$ falsifying $v'_m \vee \overline{v'_i}$ and satisfying the rest of the clauses of F, the stuck-at-1 fault ϕ is **untestable** (i.e. the input/output behavior of N does not change in the presence of ϕ).

6.4 Extracting a tight sufficient test set from a "natural" resolution proof

A "natural" proof R_{nat} that F is unsatisfiable is to derive clauses describing functional equivalence of corresponding internal points of N' and N''. These clauses are derived in topological order. First, the clauses describing the equivalence of outputs of corresponding gates of topological level 1 (whose inputs are inputs of N' and N'') are derived. Then using the equivalence clauses relating outputs of gates of topological level 1, the equivalence clauses relating outputs of corresponding gates of level 2 are derived and so on.

When building R_{nat}, we resolve clauses $F(G'_i(v'_m,v'_k))$ and $F(G''_i(v''_m,v''_k))$ describing corresponding gates G'_i and G''_i of N' and N'' and equivalence clauses $EQ(v'_m,v''_m)$, $EQ(v'_k,v''_k)$ relating inputs of G'_i and G''_i. Here $EQ(v'_m,v''_m)=(v'_m \vee \overline{v''_m}) \wedge (\overline{v'_m} \vee v''_m)$ if v'_m and v''_m are *internal* variables. If v'_m and v''_m are *input* variables of N' and N'', they denote the same input variable and $EQ(v'_m,v''_m) \equiv 1$. By resolving clauses of $F(G'_i(v'_m,v'_k)) \wedge F(G''_i(v''_m,v''_k)) \wedge EQ(v'_m,v''_m) \wedge EQ(v'_k,v''_k)$ we generate new equivalence clauses $EQ(v'_i,v''_i)$ relating the outputs of G'_i and G''_i. Let $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ be a tight point image of the resolution operation over clauses C_1 and C_2 performed when deriving clauses of $EQ(v'_i,v''_i)$. Let, say C_1, be a clause of $F(G'_i)$, $\boldsymbol{p}_1$ falsify C_1 and satisfy $F \setminus \{C_1\}$. Then, using the reasoning applied in the previous subsection, one can show that $inp(\boldsymbol{p}_1)$ is a circuit test detecting the stuck-at-fault corresponding to disappearance of C_1 from F. More detailed description of building a tight point image of R and its relation to stuck-at fault tests is given in [2]. In particular, we show that the set $inp(T_{nat})$ where T_{nat} is a tight point image of R_{nat} contains tests detecting all testable stuck-at faults. On the other hand, $inp(T_{nat})$ may have to contain tests that detect the same stuck-at-fault in different ways. So, $inp(T_{nat})$ is stronger than a test set detecting testable all stuck-at faults. Interestingly, the high quality of test sets detecting every stuck-at fault *many times* was observed in [8] experimentally.

6.5 Brief discussion

The size of R_{nat} and hence the size of T_{nat} is linear in the size of N. Moreover, since different points of T_{nat} may have identical input parts, the size of $inp(T_{nat})$ may be considerably smaller than that of T_{nat}. Importantly, T_{nat} is not meant to detect stuck-at or any other type of faults. The fact that T_{nat} does contain such tests suggests that tight test sets extracted from resolution proofs can be successfully used in manufacturing testing.

One can always get a *stronger* test set (that detects more faults of various kinds) by "rarefying" the proof R_{nat}. Suppose, for example, that a single-output subcircuit K of circuit N is particularly prone to faults and requires some extra testing. This can be achieved, for example, by dropping all the resolvents of R_{nat} that were generated from clauses $F_{K'}$ and $F_{K''}$ when obtaining the equivalence clauses $EQ(v'_i,v''_i)$. Here $EQ(v'_i,v''_i)$ relate the outputs of K' and K'' in N' and N'' and F_K are the clauses specifying the functionality of subcircuit K. Let C be a clause of $EQ(v'_i,v''_i)$. Then an SSP S of the subformula F_C (here F_C is the CNF formula built to check if F implies C) will contain more points then the part of a point image of R_{nat} corresponding to resolution operations over clauses of $F_{K'}$ and $F_{K''}$. So a test set containing S will provide better testing of the subcircuit K.

7 Experimental Results

In this section, we describe application of tight sufficient test sets to detect a change in the functionality of a combinational circuit. Such a change may be caused either by a manufacturing fault or by circuit re-synthesis.

In the experiments we compared the quality of circuit tests (i.e. complete assignments to input variables) generated randomly and extracted from tight sufficient test sets. Given a circuit N, a tight sufficient test set T was extracted from a resolution proof R that a CNF formula F describing equivalence checking of two copies of N is unsatisfiable. (The exact procedure for obtaining T from R and many more experimental results are given in [2]. As we mentioned above a resolution proof that two copies of N are functionally equivalent can be easily generated manually. However, for the sake of generality, in experiments we used resolution proofs generated by a SAT-solver, namely by the SAT-solver *FI* [3].) To form a circuit test set from T we randomly picked *a subset* of the set $inp(T)$ (where $inp(T)$ consists of the input parts of the points from T).

Table 1 shows experimental results for four circuits of a MCNC benchmark set. All circuits consist of two-input AND and OR gates inputs of which may be negated. The columns 2-4 give the number of inputs, outputs and gates of a circuit. The fifth column shows the size of the proof R (in the number of resolution operations) that two copies of circuit N are equivalent. The last column gives the size of a tight point image of R (in the number of points).

Name	#inp	#out	#gates	#proof	#point image T
c432	36	7	215	10,921	5,407
c499	41	32	414	59,582	27,903
cordic	23	2	93	1,443	808
i2	201	1	233	1,777	1,435

Table 1. The size of circuits, proofs and point images

Let F be a CNF formula describing equivalence checking of two copies N' and N'' of a circuit N. Here $F = F_M \wedge z$ where z is the variable describing the output of the miter M of N' and N'' (as shown in Fig. 3).

The fault we used in experiments was to add a literal to a clause of F_M. This fault is more subtle than a stuck-at fault in which an entire clause is removed from F_M. In [2] we give the interpretation of the literal appearance fault from a technological point of view. Literal appearance in a clause of F_M can be also used to simulate small design changes that are hard to detect in functional verification.

Let $\boldsymbol{s}$ be a circuit test (i.e. an assignment to the input variables of N). To check if ϕ is detected by $\boldsymbol{s}$ we make the assignments specified by $\boldsymbol{s}$ in F_M and run Boolean Constraint Propagation (BCP) for F_M. If z gets assigned 1 (or 0) during BCP, then $\boldsymbol{s}$ detects (respectively does not detect) ϕ.

In general however, running BCP may not result in deducing the value of z. The reason is that after adding a literal to a clause of F_M, circuit behavior becomes non-deterministic. For example, let $C = \overline{v'_i} \vee \overline{v'_j} \vee v'_k$ be a clause of the CNF $F(G'_k)$ describing the functionality of the AND gate $G'_k(v'_i, v'_j)$. Suppose that ϕ is to add literal v'_m to C. Normally, if v'_i=1,v'_j=1, the value v'_k=1 is derived from the clause C. However, if the value of v'_m becomes equal to 1 during BCP (before the variable v'_k is assigned), then the clause $\overline{v'_i} \vee \overline{v'_j} \vee v'_k \vee v'_m$ is satisfied without assigning 1 to v'_k. So the output of the gate G'_k remains unspecified under the input assignment $\boldsymbol{s}$. In this case, we run a SAT-solver trying to assign values to the unassigned variables to satisfy F (and so set z to 1). If such an assignment exists (does not exist), $\boldsymbol{s}$ is considered to detect (not to detect) ϕ. The reason is that if ϕ simulates a manufacturing fault and we succeed in satisfying the faulty F, then $\boldsymbol{s}$ will detect ϕ in case the output of G'_k is set to the wrong value (i.e. 0).

Name	#tests	SIS #flts	rand #flts	extr. from *inp*(T) #flts
c432	58	86	69.7(65)	79.7 (76)
	100	-	77.1 (72)	86.7 (78)
	200	-	88.7(85)	95.5 (90)
c499	93	90	78.7 (70)	85.9(83)
	200	-	86.9 (84)	91.2 (89)
	400	-	91 (88)	95.2 (92)
cordic	43	84	28.5 (23)	81.6 (74)
	100	-	36.6 (29)	94.2 (87)
	200	-	54.8 (36)	99 (98)
i2	221	71	7.8 (3)	66.4 (62)
	400	-	9.2 (6)	74.6 (69)
	600	-	11.6 (10)	82.4 (80)

Table 2. Circuit testing

Table 2 shows the results of fault testing for the circuits of Table 1. In every experiment we generated 100 *testable* faults (i.e. every fault specified a satisfiable variation of F). The second column of Table 2 gives the size of a test set. The third column gives the result for a test set detecting all stuck-at faults in N. This test set was generated by the logic synthesis system SIS [7]. Since we could not vary the size of the test set produced by SIS, only one test set was used per circuit. For example, for the circuit *c432*, a test set of 58 tests was generated by SIS. These tests were able to detect 86 out of 100 faults of literal appearance. The fourth column contains the results of fault detection using circuit tests generated randomly. In every experiment we used 10 test sets and computed the average result.The value in parentheses shows the worst result out of 10. For example, for the circuit *c432*, in the first experiment (first line of Table 2) we generated 10 random test sets, each consisting of 58 tests. On average, 69.7 faults were detected, 65 faults being the worst result out of 10.

The fifth column contains the result of fault detection using circuit tests extracted from the set *inp*(T) where T is a point image of a proof R that F is unsatisfiable. Namely, we *randomly* extracted a particular number of tests from *inp*(T). The corresponding sizes of T are given in Table 1. In every experiment we also generated 10 test sets of a particular size and we give the average value and the worst result out of 10. For example, in the first experiment, for the circuit c432, 10 test sets of 58 tests each were extracted from *inp*(T). The average number of detected faults was 79.7 and the worst result was 76 faults.

Table 2 shows that tests extracted from a point image T of a resolution proof R perform better than random tests. For circuits *c432*, *c499* that are shallow (i.e. have few levels of logic) and have relatively large number of outputs (7 and 32 respectively) tests extracted from resolution proofs performed only slightly better. (Testing shallow circuits with many outputs is easy). However, for circuits *cordic* and *i2* that are also shallow but have only 2 and 1 outputs respectively tests extracted from resolution proofs significantly outperformed random tests.

Table 2 also shows that the quality of a test set extracted from a resolution proof depends on proof quality. As we mentioned above, tests detecting stuck-at faults is a part of $inp(T_{nat})$ where T_{nat} is a point image of a natural resolution proof R_{nat}. Table 2 shows that tests found by SIS performed better than tests extracted from proofs found by FI (these proofs are significantly larger than R_{nat}).

8 Conclusion

In this paper, we develop a theory of sufficient test sets. The essence of our approach is to interpret a set of tests not as a sample of the search space but as an encoding of a formal proof. We believe that this theory can have many applications. An obvious application is generation of high-quality tests. We show that such tests can be extracted from resolution proofs (possibly rarefied). One more interesting direction for research is extending the notion of stable sets of points (which is the foundation of our approach) to domains other than propositional logic. This may lead to developing new methods of generating high quality test sets for more complex objects like sequential circuits or even programs.

References

1. M. Abramovici, M.A.Breuer, A.D.Friedman. *Digital Systems Testing and Testable Design.* 672 p.Sep. 1994, Wiley-IEEE Press
2. E.Goldberg. *On bridging simulation and formal verification.* Technical Report CDNL-TR-2006-1225, December 2006 (available at http://eigold.tripod.com/papers/ssim.pdf).
3. E.Goldberg. *Determinization of resolution by an algorithm operating on complete assignments.* SAT-2006, LNCS 4121, pp.90-95.
4. E. Goldberg. *Testing Satisfiability of CNF Formulas by Computing a Stable Set of Points*, CADE 2002, LNCS, vol 2392,pp.161-180.
5. B. Selman H. Levesque, D. Mitchell. 1992. *A New Method for Solving Hard Satisfiability Problems.* AAAI-92, pp. 440-446.
6. B.Selman, H.A.Kautz. and B.Cohen. *Noise strategies for improving local search.* AAAI-94, Seattle, pp. 337-343, 1994.
7. E.Sentovich et. al. SIS: A system for sequential circuit synthesis. Technical report, University of California at Berkeley, 1992. Memorandum No. UCB/ERL M92/41
8. E. McCluskey and C. Tseng, *Stuck-fault tests vs. actual defects*, Proc. of Int. Test Conf., pp. 336 -343, 2000.
9. L.Bachmair, H.Ganzinger. A.Robinson, A.Voronkov editors, The Handbook of Automated Reasoning, chapter 2, vol. 1,19-99. Elsevier Science Pub.2001.