On Bridging Simulation and Formal Verification

Eugene Goldberg

Cadence Berkeley Labs, USA, 1995 University Ave., Suite 460, Berkeley, California, 94704, phone: (510)-647-2825, fax: (510)-486-0205, `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 a "prover" that can rigorously prove unsatisfiability of CNF formulas. We introduce the notion of a *sufficient* test set of a CNF formula which is a test set that has "enough power" to prove the unsatisfiability of this formula. We show how sufficient test sets can be built. We discuss applications of *"tight"* sufficient test sets in manufacturing testing and 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 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 a *prover* that can rigorously prove the unsatisfiability of CNF formulas.

We introduce a procedure $Sat(T,F,L)$ that proves unsatisfiability of a CNF formula F using a test set T as a "prover". Here L is a set of "lemma" clauses (or just lemmas for short). If for some point $\boldsymbol{p} \in T$, $F(\boldsymbol{p})$ is equal to 1, $Sat(T,F,L)$ reports that F is satisfiable. Otherwise, $Sat(T,F,L)$ either proves that F is unsatisfiable or reports failure. 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 set of lemma clauses $L_1,\ldots,L_k$ is ordered and the last clause L_k is empty. The procedure $Sat(T,F,L)$ takes a clause L_i and checks if F implies L_i. If it succeeds in proving this implication, L_i is *added* to F. (Otherwise, $Sat(T,F,L)$ reports failure.) Then $Sat(T,F,L)$ starts processing clause L_{i+1}. The implication check above is based on computing a stable set of points (SSP) [4]. Namely, in this paper, we describe an efficient procedure that, given a set of points T and a CNF formula F', checks if a subset of T is an SSP for F'. This procedure is used by $Sat(T,F,L)$ to check if $F \to L_i$. So, in a sense, the test set T is used by $Sat(T,F,L)$ as a "prover".

The fewer lemmas a sufficient test set T "needs" for proving unsatisfiability of F by $Sat(T,F,L)$, 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 the most powerful "prover". 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, in a sense, the weakest "prover".

Sufficiency of a test set T with respect to an unsatisfiable CNF formula F makes this set "complete" in some sense. However, such completeness alone does not make T good for detecting *small variations* of F. In this paper, we introduce *"tight"* sufficient test sets that are "sensitive" to formula variations and show how such test sets can be built.

Given a CNF formula F, one can build a (tight) sufficient test set T as a point image of a resolution proof R that F is unsatisfiable. One can get a stronger test set by rarifying the proof R (i.e. by removing some resolvents from R). The idea is that rarification of R makes it harder to prove $F \to C$ (where C is a remaining resolvent of R) using a test set as a prover. So one has to build a stronger test set T. In particular, if one removes from R all resolvents but an empty clause, the test set turns into an SSP of F. By varying the degree of rarification of R one can find the required trade-off between the size and the quality of T.

There are at least two areas of application of our theory. The first area is generation of manufacturing tests. In terms of SAT, the task of manufacturing testing is as follows. Given an unsatisfiable CNF formula F, one needs to find a set T of points that can detect if F becomes satisfiable after a small variation ("fault"). The second area is functional verification. In terms of SAT, functional verification is either to prove unsatisfiability of a CNF formula F or to get some "guarantee" that F is unsatisfiable. (In other words, functional verification is

the superset of formal verification and simulation.) Interestingly, in functional verification there is an application of sufficient test sets similar to the one mentioned above (i.e. finding a test set detecting if a variation makes F satisfiable). However, in the case of functional verification, a variation of F describes not a manufacturing fault but a small design change.

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. The notion of a sufficient test set can be also used to explain the success of corner case driven test generation. Currently, tests exercising design "corner cases" is one of the key contributors to the good performance of simulation. Intuitively, these kind of tests are most likely to be a part of a *sufficient* test set. We substantiate these intuition in Section 6. We show that a tight sufficient test set extracted from a "natural" resolution proof that two copies of a circuit are functionally equivalent contains all the tests for detecting stuck-at faults[1]. On the one hand, such tests are ubiquitous in circuit testing. On the other hand, they are exactly aimed at exercising corner cases of circuit behavior.

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. Sections 6 and 7 describe application of sufficient test sets in manufacturing testing and functional verification. We give some experimental results in Section 8 and conclude by Section 9.

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.

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 $Vars(C)$ and $Vars(F)$ 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.

Stable set of points. Let a point $\boldsymbol{p} \in \{0,1\}^{|X|}$ falsify a clause C of k literals. Denote by $Nbhd(\boldsymbol{p},C)$ 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

$C = x_2 \lor x_3 \lor \overline{x_5}$ 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 $Nbhd(\boldsymbol{p},C) = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3\}$ where $\boldsymbol{p}_1 = (.., x_2=1,..)$, $\boldsymbol{p}_2=(.., x_3=1,..)$, $\boldsymbol{p}_3 = (\ldots, x_5=0)$. (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 $C_1,\ldots,C_s$. Let $T = \{\boldsymbol{p}_1,\ldots,\boldsymbol{p}_m\}$ be a non-empty set of points from $\{0,1\}^{|X|}$ such that $F(\boldsymbol{p}_i)=0$, $i=1,..,m$. Let $g: T \rightarrow F$ be a function mapping each point $\boldsymbol{p}_i$ of T to a clause C_j of F such that the clause $C_j = g(\boldsymbol{p}_i)$ is falsified by $\boldsymbol{p}_i$. The set T is called a **stable set of points** (**SSP**) of F with respect to a function g, if for each $\boldsymbol{p}_i \in T$, $Nbhd(\boldsymbol{p}_i,g(\boldsymbol{p}_i)) \subseteq T$.

Proposition [4]. 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 $\{0,1\}^{|X|}$ and a function $g: T \rightarrow F$ such that T is an SSP with respect to g.

Checking if test set contains SSP. Given a set of points T and a CNF formula F, the procedure of **Figure 1** checks if there is subset of T that is an SSP of F with respect to some function g. For every point $\boldsymbol{p}$ of T it checks

```
Stable_subset_check(T,F)
  {removed=true;
  while (removed)
    {removed=false;
    for (every point p ∈ T)
      if (no_clause(p,F,T))
        {T = T\ {p};
        removed=true;
        break;}}
  if (T ≠ ∅) return (stable)
  else return(unstable);}
```

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. Even if in the previous iterations a point $\boldsymbol{p}^*$ was not removed from T, after removing $\boldsymbol{p}$, the situation may change for $\boldsymbol{p}^*$.) This repeats until no point is removed from T, which may happen only in two cases. a) T is empty (and so it does not contain a stable subset). b) The remaining points

Figure 1. Checking if T contains an SSP

of T form an SSP. The complexity of this procedure is $|T|^2*|F|*|X|$. On the other hand, as we show in [2] (a more detailed version of the current paper) the complexity of the procedure that checks if T *itself* is an SSP is $|T|*|F|*|X|$.

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. We also introduce the notion of a sufficient test set and describe how sufficient test sets can be obtained.

$SAT(T,F,L)$ 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

```
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);}
```

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

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

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**.)

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 "power" 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.

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''.

Let C be the resolvent of C' and C''. Let the set L of lemma clauses consist of the resolvents of R. 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'' have been already added to F. Let F_C be the formula obtained from F for checking if F implies C (by making the assignments setting the literals of C to 0). 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 if L consists of the resolvents of a proof R and T is a point image of R. A point image is the "weakest" sufficient test set, because it is able to prove only the weakest lemmas (that the resolvent of C' and C'' is implied by $C' \wedge C''$).

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). Note that 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 the "weakest" sufficient test set. However, one can always get a "stronger" test set by "rarifying" R. The idea is to remove some resolvents from R 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 $L_1,..,L_{i-1}$ "proven" 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 some sense. However, this completeness alone does not make T a high-quality test set. In practical applications, one needs to generate test sets that are "sensitive" to small variations F that make it satisfiable. So, 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). 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 $\boldsymbol{p}$).

One can give a more "high-level" explanation of why a sufficient test set T of F should be tight. In general, we want T to be unique for F and so "unusable" by other CNF formulas. Then if F changes, T will be either insufficient (if F remains unsatisfiable) or with great probability will contain a satisfying assignment (if F becomes satisfiable).

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 subformula 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. In this version, in points $\boldsymbol{p'},\boldsymbol{p''}$, only the parts consisting of the assignments of the variables of $Vars(C') \cup Vars(C'')$ 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 one can build a tight sufficient test set extracted from a "rarified" resolution proof (that was introduced in subsection "Generation of sufficient test sets" of Section 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*. We discuss some specifics of testing formulas of Circuit-SAT.

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**.

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 an instance of *Circuit-SAT*. 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 $T= \{\boldsymbol{p_1},\ldots,\boldsymbol{p_m}\}$, one can always form a circuit test set $inp(T)= \{\boldsymbol{y_1},\ldots,\boldsymbol{y_k}\}$, $k \leq m$, 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 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 if one applies the assignments of $inp(\boldsymbol{p_i})$ to the input variables of the changed circuit, the latter evaluates to 1. In other words, the "internal" part of $\boldsymbol{p_i}$ describes what change ("fault") needs to be brought into circuit N to make $inp(\boldsymbol{p_i})$ a circuit test that "detects" that N became satisfiable.

6   Manufacturing Testing

We showed above how a tight sufficient test set can be built from a resolution proof (possibly "rarified"). In this section, we describe how one can use tight sufficient test sets for manufacturing testing. In terms of SAT, the objective of manufacturing testing is to detect a variation ("fault") of an unsatisfiable F that makes the latter satisfiable. Usually, to reduce the size of test set, a fault model (e.g. the stuck-at fault model [1]) is specified. 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 a 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.

In this section, we show how one can use tight sufficient test sets for manufacturing testing of a circuit N. The idea is to build a resolution proof R that a property ξ of N holds and then use R (possibly "rarified") to build a *tight* sufficient test set T. This test set is meant to detect faults that break the property ξ. Importantly, such a test set is *fault model independent*. Every point $\boldsymbol{p_i}$ of T can be trivially transformed to a circuit test by taking the input part of $\boldsymbol{p_i}$.

The most "fundamental" property of a circuit is 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 show 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.) This result offers 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 describing 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.

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).
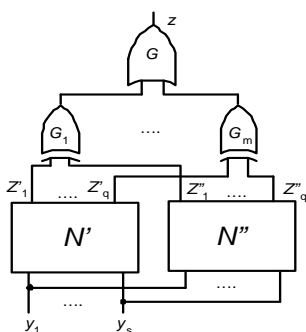

Fig. 3. Miter M of circuits N' and N''

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 **Figure 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.

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 constant value.) 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 $F(G_i)$.

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 ϕ).

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 at 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 we 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, $inp(T_{inp})$ may have to contain tests that detect the same stuck-at-fault "in different ways". So, $inp(T_{inp})$ is stronger than a test set detecting testable all stuck-at faults.

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 indicates 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 "rarifying" the proof R. Suppose, for example, that a 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 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 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 Functional Verification

At first glance, building a sufficient test set based on a proof that a formula F is unsatisfiable does not make sense in functional verification (i.e. checking the unsatisfiability of F either formally or by simulation). Indeed, if we have a proof that F is unsatisfiable, there is no need to test F by "simulation". However, as we show below, there are scenarios under which tight sufficient test sets can be used to generate high-quality test sets. Here we assume that a CNF formula F specifies some property ξ of a circuit N. Unsatisfiability of F means that ξ holds for this circuit. One more assumption is that proving the unsatisfiability of F is hard or takes too much time and we would like to test ξ by simulation. We sketch two (out of many) ways to improve the quality of this simulation that are based on application of our theory of sufficient tests sets.

Testing a modified circuit. Suppose we managed to generate a resolution proof R that ξ holds. Suppose we have to make a small change in N and would like to know if the property ξ holds for the new circuit N'. Then instead of generating a new proof we can use R to build a tight sufficient test set T and then apply the input parts of points of T to test N'.

Testing property ξ under assumptions. Suppose that we can not prove the unsatisfiability of F but succeeded in proving the unsatisfiability of $F \wedge H$ where H is a CNF formula consisting of *assumption clauses*. Then the property ξ is proved to hold for every point $\boldsymbol{p}$ for which $H(\boldsymbol{p}) = 1$. A trivial way to use this fact is to "randomly" generate points $\boldsymbol{p}$ for which $H(\boldsymbol{p})=0$ trying to find one for which F evaluates to 1. (Generation of "random" assignments that satisfy some constraints is a widely-used technique now.) However one can do better by making use of the resolution proof R found when proving unsatisfiability $F \wedge H$. The idea is to build a tight point image T of R of a special kind. When generating a point $\boldsymbol{p}$ of T it is allowed to falsify an *arbitrary* number of clauses of H. At the same time the number of clauses of F falsified by $\boldsymbol{p}$ is supposed to be kept to minimum. If a point satisfying all the clauses of F is found, then ξ does not hold. The difference of this approach from just looking for a point for which $H(\boldsymbol{p})=0$ and $F(\boldsymbol{p})=1$ is as follows. When building a point image of a resolution operation some value assignments of $\boldsymbol{p}$ are mandatory. So the search of a counterexample

$\boldsymbol{p}$ is "guided" by resolutions of R. This search is "complete" in the sense that when "attacking" (i.e. falsifying) assumptions of H we take into account every situation in which an assumption clause (or its descendant) was used in R.

8 Experimental Results

In this section, we describe application of tight sufficient test sets to circuit testing. 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]. Resolution proofs were generated 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

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

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.)

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 Figure 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. Although the literal appearance fault does not exactly correspond to an existing model of a manufacturing fault, in [2] we give its interpretation 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 adding a literal to a clause of F_M is not a "functional fault".

For example, let the $C = \overline{v'_i} \vee \overline{v'_j} \vee v'_k$ 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.

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 made F satisfiable). 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 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.

One can draw at least three conclusions from Table 2. First, 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 these tests performed better than tests extracted from proofs found by *FI* (that are significantly larger). Second, even though tests detecting the stuck-at faults have high quality, they do not detect all literal appearance faults. Third, 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 of 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.

9   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 a formal "prover". We believe that this theory can have many applications. One obvious application is generation of high-quality tests. We show that such tests can be extracted from resolution proofs. 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