On Bridging Simulation and Formal Verification

Cadence Berkeley Labs
1995 University Ave., Suite 460, Berkeley, California,94704
phone: (510)-647-2825, fax: (510)-486-0205

CDNL-TR-2007-0123
January 2007

Eugene Goldberg (Cadence Berkeley Labs), 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 a CNF formula. 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.**

I. 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 property ξ to be checked, 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 that it is unscalable while an obvious flaw of simulation is that it can not guarantee that ξ holds for every point of the search space.

Nevertheless, the main bulk of verification is currently done by simulation for the following two reasons. First, simulation is scalable. Second, simulation works surprisingly well taking into account the negligible part of the search space covered by a set of test points (further referred to as a **test set**).

We study the phenomenon of the effectiveness of simulation by 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 $\{0,1\}^n$. (In a sense, the local search algorithms pioneered in [5][6] can be viewed as solving SAT by "simulation". In contrast to local search algorithms that target *satisfiable* formulas, in this report, 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) [3][4]. Namely, in this report, 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 $F \rightarrow L_i$. 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 [2], we introduced the notion of a point image of a resolution proof R that a CNF formula is unsatisfiable. In this report, we show 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 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 if a small variation of F makes the latter satisfiable. In this report, 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. In this report, we describe how 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 that $F \rightarrow C$ (where C is a remaining resolvent of R) using a test set 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, T 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 the negligible part of the search space. (For example, a point image of a resolution proof is two times the size of the proof at most.) This fact sheds some 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 corner cases of design behavior is one of the key contributors to the good performance of simulation. Intuitively, this kind of tests are most likely to be a part of a *sufficient* test set. We substantiate these intuition in Section VI. 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 report is structured as follows. Section II describes procedures for checking if a set of points or its subset are SSPs with respect to a CNF formula. In Section III, 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 IV. In Section V, we discuss the specifics of testing formulas describing circuits. Sections VI and VII describe application of sufficient test sets in manufacturing testing and functional verification. We give some experimental results in Section VIII and conclude by Section IX.

II. Checking If Test Set *T* Contains SSP

In this section, we give some basic definitions, recall the notion of a stable set of points (SSP) [3][4] and introduce two procedures that check if a set of points or its subset are stable.

A. *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 $Vars(C)$, $Vars(F)$ the set of variables of clause C and CNF formula F respectively. Let F be a CNF formula and X be its set of variables. We will call a complete assignment $\boldsymbol{p} \in \{0,1\}^{|X|}$ **a test**. We will call a set of points $T \subseteq \{0,1\}^{|X|}$ **a test set**.

Let C_1 and C_2 be two clauses that have opposite literals of a variable x_m. Then the clause consisting of all the literals of C_1,C_2 except those of x_m is called the **resolvent** of C_1,C_2. (For example if C_1=$x_1 \vee x_3 \vee x_5$, C_2= $x_2 \vee \sim x_3 \vee x_7$, the resolvent of C_1 and C_2 is the clause $x_1 \vee x_5 \vee x_2 \vee x_7$.) The resolvent of C_1,C_2 is said to be obtained by the **resolution operation**. Denote by $Res(C_1,C_2)$ the resolution operation over clauses C_1 and C_2. Given an unsatisfiable CNF formula F, one can always generate a sequence of resolution operations resulting in producing an empty clause. This sequence of operations is called a **resolution proof** that formula F is unsatisfiable.

B. *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=$\{x_1,.., x_5\}$, $C = x_2 \vee x_3 \vee \sim x_5$ 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,..), $\boldsymbol{p}_3$ = (…, 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,…,C_s. Let T = $\{\boldsymbol{p}_1,…, \boldsymbol{p}_m\}$ 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 $\boldsymbol{p}_i$. (g is called **a transport function** because it defines "movement" of points in T.) The set T is called a **stable set of points** (**SSP**) of F with respect to a transport function g, if for each $\boldsymbol{p}_i \in T$, $Nbhd(\boldsymbol{p}_i, g(\boldsymbol{p}_i)) \subseteq T$.

***Proposition* 1. [3].** 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 transport function g: $T \rightarrow F$ such that T is an SSP with respect to g.

C. Checking if a test set contains an SSP

Let F be a CNF formula over a set X of Boolean variables. Let $T \subseteq \{0,1\}^{|X|}$ be a set of points such that $F(\boldsymbol{p})=0$ for every point $\boldsymbol{p}$ of T. **Figure 1** shows pseudocode of a procedure that checks if T is stable. For every point $\boldsymbol{p}$ of T, this procedure checks if there is a clause C of F such that $Nbhd(\boldsymbol{p},C) \subseteq T$. If such a clause does not exist, then T is not stable with respect to F. On the other hand, if such a clause is found for every point $\boldsymbol{p}$ of T, T is an SSP. The complexity of this procedure is $|T|*|F|*|X|$ where $|F|$ denotes the number of clauses in F. One can interpret the procedure of *Stable_set_check* as checking if a set of points T has a transport function g: $T \rightarrow F$ making T an SSP with respect to F and g.

```
Stable_set_check(T,F)
{for (every point p ∈ T)
    {found = false;
     for (every clause C ∈ F)
        {if (p falsifies C)
           if (Nbhd(p,C) ⊆ T)
              {found = true;
               break;}}
  if (not found)
     return('unstable');}
 return('stable');}
```

Figure 1. Checking if *T* is an SSP

A drawback of the procedure of **Figure 1** is that it fails to recognize the situation when T is not an SSP but a subset of T is. The procedure of **Figure 2** solves this problem. For every point $\boldsymbol{p}$ of T it checks if there is a clause C 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 of T form an SSP. The complexity of this procedure is $|T|^2*|F|*|X|$.

```
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'); }
```

Figure 2. Checking if a subset of *T* is an SSP

III. *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 unsatisfiability of a CNF formula F. We also introduce the notion of a sufficient test set and describe how sufficient test sets can be obtained.

A. Sat(T,F,L) procedure

The pseudocode of the procedure *Sat*(T,F,L) is shown in **Figure 3**. 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, then *Sat(T,F,L)* reports that F is satisfiable. Otherwise, *Sat(T,F,L)* processes the clauses of L in the order they are numbered. For every clause L_i of L , this procedure checks if F implies L_i, by calling the function *implies*(T,F,L_i). If this function succeeds in proving the 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 2 as follows. First, the subformula F_{Li} is obtained from F by making the assignments setting all the literals of L_i to 0. Formula F implies L_i iff F_{Li} unsatisfiable. To check if F_{Li} is unsatisfiable, the procedure *Stable_subset_check*(T_{Li},F_{Li}) is called by the function *implies*(T,F,L_i) where T_{Li} is the subset of points of T falsifying L_i. This procedure checks if the set T_{Li} contains a subset that is an SSP with respect to F_{Li}. 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. However, as we will show in Subsection IV.A, this complexity can be reduced to linear in $|T|$, if some additional information is provided for *Sat*(T,F,L).

```
// we assume that the last clause
// of A is empty
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 3. Pseudocode of the procedure *Sat*(*T*,*F*,*L*)

B. Sufficient test sets

We will say that a test set T is **sufficient,** if there is a set L of lemma clauses for which *Sat*(T,F,L) succeeds in proving the unsatisfiability of F. In other words, 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 L has, the larger set T of points is necessary for *Sat*(T,F,L) to succeed. In particular, if L consists only of an empty clause, *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 set of points T is called a **point image of *R*** [2] if for any resolution operation $Res(C',C'')$ of R over clauses, 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 *Res*(*C′*,*C″*)**.

Let C be the resolvent of C' and C''. Let the set L of lemma clauses used by *Sat*(T,F,L) consist of the resolvents of R. Then C is in L. When *Sat*(T,F,L) gets to proving that C is implied by the current formula F, clauses C' and C'' are already in F.

Let F_C be the formula obtained from F by making the assignments setting the literals of C to 0 (i.e. F_C is the formula used for checking if F implies C). Clauses C' and C'' turn into unit clauses x_i and $\sim x_i$ in F_C (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. In a sense, 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 $C' \wedge C''$).

C. Generation of sufficient test sets

The fact that a point image of a resolution proof is a sufficient test set can be used for building such test sets. Given a proof R that a CNF formula F is unsatisfiable, building a point image of R 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 resolution $Res(C',C'')$. The union of point images of all the resolution operations forms a point image of R (and hence a sufficient test set). 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 of the CNF formula F_{Li} (obtained from F by making the assignments falsifying the literals of L_i) thus proving that $F \rightarrow L_i$. (Here, 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 [3][4]. Since some resolvents 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.

IV. 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. 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 F as possible. (Ideally, every point $\boldsymbol{p}$ of T should falsify only one clause of F). 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 a "unique" proof that F is unsatisfiable and so "unusable" by other CNF formulas. Then, if F changes, T is either insufficient (if F is unsatisfiable) or with great probability contains a satisfying assignment (if F is satisfiable).

Before describing in Subsection B how to build tight sufficient tests, in Subsection A, we give a description of two modifications of the procedure $Sat(T,F,L)$. These modifications improve its performance and relax the definition of a sufficient test set.

A. Modified SAT(T,F,L)

Denote by $Sat^*(T,F,L,A)$ the procedure that has the following two modifications of the procedure $Sat(T,F,L)$ introduced in Subsection III.A. The first modification is that in addition to the set L of lemma clauses, $Sat^*(T,F,L,A)$ also gets information represented by parameter A. Namely, for every lemma L_i, Sat^* gets information about a subset T_i of T that is an SSP of the subformula F_{Li} formed to check if F implies L_i. (We assume that F contains the lemma clauses $L_1,..,L_{i-1}$). Sat^* also gets information about the set of variables $Vars(F'_{Li})$ where F'_{Li} is a set of clauses of F_{Li} that is sufficient to examine when checking if T_i is an SSP of F_{Li}. Since the set of points forming an SSP is known, Sat^* uses procedure *Stable_set_check* of **Figure 1** (instead of *Stable_subset_check*). So in contrast to $Sat(T,F,L)$, the complexity of Sat^* is $|T|*|F|*|X|*|L|$ i.e. linear in $|T|$.

The fact that Sat^* "knows" the set $Vars(F'_{Li})$ allows one to make one more modification of $Sat(T,F,L)$. This modification is to change the *Stable_set_check* procedure in the following way. Before checking by the *Stable_set_check* procedure, if the points of T_i form an SSP with respect to F_{Li}, both T_i and F_{LI} are "projected" to the set of variables $Vars(F'_{Li})$. That is from all the points of T_i, the assignments to the variables that are not in $Vars(F'_{Li})$ are removed. All the clauses of F_{Li} having variables that are not in $Vars(F'_{Li})$ are dropped. This modification of $Sat(T,F,L)$ "relaxes" the definition of a sufficient test set. Denote by $Proj(T_i)$ and $Proj(F_{Li})$, set T_i and formula F_{Li} projected as described above. Even if T_i is not an SSP with respect to F_{Li} , $Proj(T_i)$ may be an SSP with respect to $Proj(F_{Li})$. Note that this projection is a sound operation. Indeed, let D be an assignment the variables of F that are not in $Vars(F'_{Li})$. Let T'_i be the set of points that are obtained from $Proj(T_i)$ by extending every point of $Proj(T_i)$ with the same assignment D. Then, if $Proj(T_i)$ is an SSP for $Proj(F_{Li})$, the set T'_i is an SSP for F_{Li}.

As we will see in the next subsection, the relaxation above is very useful when building *tight* sufficient test sets.

B. Building tight sufficient test sets

Let us consider building a tight test set T sufficient with respect the procedure Sat^* above. Let T be a point image of a resolution proof R. Then the set L of lemmas consists of the resolvents of R. Let L_i of L be the resolvent of R over clauses C' and C''. Since procedure Sat^* is used for checking if T is a sufficient test set, the information about the set of variables $Vars(C') \cup Vars(C'')$ is passed to Sat^* (because these are the clauses forming an unsatisfiable core of F_{Li}). When looking for two points $\boldsymbol{p}'$ and $\boldsymbol{p}''$ forming a point image of the resolution $Res(C',C'')$ (and so forming an SSP of subformula F_{Li}) we have freedom in assigning. the variables of F that are not in $Vars(C') \cup Vars(C'')$. To make the test set T tight, these assignments should be chosen to minimize the number of clauses falsified by $\boldsymbol{p}'$ and $\boldsymbol{p}''$. Note that the *Stable_set_check* procedure changed as described above, will drop the assignments to variables that are not in $Vars(C') \cup Vars(C'')$. So points $\boldsymbol{p}'$ and $\boldsymbol{p}''$ do not have to be at distance 1. Only the parts of $\boldsymbol{p}'$ and $\boldsymbol{p}''$ consisting of the variables of clauses C' and C'' should be at distance 1. To satisfy the latter condition it is sufficient to require that both $\boldsymbol{p}'$ and $\boldsymbol{p}''$ that falsify the resolvent of C' and C''. Summarizing we will say that points $\boldsymbol{p}'$ and $\boldsymbol{p}''$ form a **relaxed point image** of $Res(C',C'')$ if a) $C'(\boldsymbol{p}')=0$ and $C''(\boldsymbol{p}'')=0$; b) $C(\boldsymbol{p}')=C(\boldsymbol{p}'') = 0$ where C is the resolvent of C' and C''.

Suppose $L = \{L_1,..,L_k\}$ is not the set of resolvents of a proof (e.g. L may be a rarified resolution proof (Subsection III.C)). Then a tight sufficient test set can be built in the following way. Let L_i be a lemma clause and S_i be an SSP for the formula F_{Li} with respect to a transport function g_i. (We assume that F contains the lemma clauses $L_1,..,L_{i-1}$). Denote by F'_{Li} the subset of clauses of F_{Li} assigned to points of S_i by the transport function g_i. When forming a tight sufficient test, one has the freedom in choosing assignments to the variables of F that are not in $Vars(F'_{Li})$. So, for every point $\boldsymbol{p}$ of S_i, we try to assign the variables of $Vars(F) \setminus Vars(F'_{Li})$ to minimize the number of clauses of F falsified by $\boldsymbol{p}$. The set $S_1 \cup \ldots \cup S_k$ forms a tight sufficient test set "specified" by the set of lemma clauses L.

V. 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 formulas of Circuit-SAT.

A. Circuit-SAT

Let N be a single-output combinational circuit. Let F_N be a CNF formula obtained from N as usual. 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 $F_N = F(G_1) \wedge .. \wedge F(G_k)$. 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 $(\sim v_m \vee \sim v_n \vee v_i) \wedge (v_m \vee \sim v_i) \wedge (v_n \vee \sim v_i)$. 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**.

B. 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 corresponding 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 the input of $\boldsymbol{p}$ 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, working with cubes this large is unacceptable because the complexity of $Sat(T,F,L)$ is proportional to the size of T.

Note that, given a sufficient test set $T=\{\boldsymbol{p_1},..,\boldsymbol{p_m}\}$, one can always form a test set $inp(T)=\{\boldsymbol{y_1},..,\boldsymbol{y_k}\}$, $k \leq m$, consisting of input parts of the points from T. (Some points of T may have identical input parts and 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 in $\boldsymbol{p_i}$ is equal to 1. 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.

VI. 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 know what kind of faults may occur. 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. Then test T can be used to detect faults that break the property ξ. Importantly, such a test set is *fault model independent*. Every test $\boldsymbol{p}_i$ 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 "natural" tight sufficient test set for the formula specifying self-equivalence of N contains *all the tests* detecting 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 in this section is structured as follows. First we introduce a circuit called a miter, that is used for equivalence checking. Then we give the definition of a stuck-at fault. 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 T_{nat} of a "natural" resolution proof R_{nat} that F is unsatisfiable is, in general, stronger than a test set detecting all (testable) stuck-at fault. Namely, $inp(T_{nat})$ contains tests detecting all (testable) stuck-at faults and some "other" tests.

A. Manufacturing tests and self-equivalence check

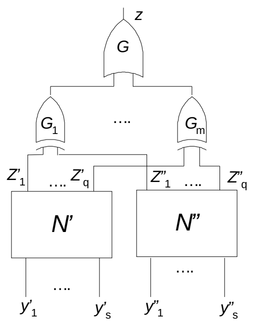

Figure 4 shows a circuit M (called **a miter**) composed of two s-input, q-output circuits N' and N''. The miter of N' and N'' is a circuit commonly used to check the equivalence of N' and N'' . Denote $EQ(y'_k,y''_k)$ the Boolean function that is equal to

Figure 4. Miter M of circuits N' and N''

1 iff $y'_k=y''_k$. The function $EQ(y'_k,y''_k)$ can be described by the CNF formula $= (\sim y'_k \vee y'') \wedge (y'_k \vee \sim y'')$. Circuits N' and N'' are functionally equivalent iff the CNF formula $F=F_M \wedge EQ(\boldsymbol{y}',\boldsymbol{y}'') \wedge z$ is unsatisfiable. (Here F_M specifies the functionality of M, $EQ(\boldsymbol{y}',\boldsymbol{y}'')= EQ(y'_1,y''_1) \wedge \ldots \wedge EQ(y'_s, y''_s)$ and z is the output variable of M.) Indeed, F is satisfied only if there is an input assignment $\boldsymbol{y}=\boldsymbol{y}'=\boldsymbol{y}''$ (so $EQ(\boldsymbol{y}',\boldsymbol{y}'')=1$) for which miter M evaluates to 1 (and so z=1). But M evaluates to 1 iff N' and N'' produce different output assignments. (Note that in the traditional definition of a miter, N' and N'' have the same set of variables $y_1,..,y_s$ and so one does not have to introduce the CNF formula $EQ(\boldsymbol{y}',\boldsymbol{y}'')$. The reason for using separate input variables in N' and N'' is given in Remark 2 of the Appendix.)

Denote by F^*_M the formula $F_M \wedge EQ(\boldsymbol{y}',\boldsymbol{y}'')$. 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 4** are just copies of N with separate sets of 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 since T is tight it can be used for detection of variations of F describing appearance of a fault in one of the copies of N.

B. Stuck-at faults

A stuck-at fault in a circuit N, describes the situation when a line in N is stuck at a *constant* value 0 or 1. Let $G_i(v_k,v_m)$ be a gate of N. Figure 5 shows examples of stuck-at faults. For example, the appearance of a stuck-at-1 fault ϕ on the input line s of G_i, means that for every assignment to the inputs of N the value of this line is 1. Note that although all three lines s,d,f are described by the same variable v_k stuck-at faults on s,d and f are *different* faults. Henceforth, when we say that there is a stuck-at fault on the input line v_k of G_i (or G_p) we mean a stuck-at fault on line s (respectively line d). When we say that there is a fault on the output line v_k of G_k we mean the line f.

Let G_i be an AND gate. Then the functionality of G_i can be described by the CNF formula $F(G_i)$=$\sim v_k \vee \sim v_m \vee v_i$, $v_k \vee \sim v_i$, $v_m \vee \sim v_i$ where v_i describes the output of G_i. The stuck-at-1 fault ϕ on the input line v_k of G_i, can be simulated by removing the clause $v_k \vee \sim v_i$ from $F(G_i)$ (it is satisfied by v_k=1) and by removing the literal $\sim v_k$ from $\sim v_k \vee \sim v_m \vee v_i$ of $F(G_i)$. (Note that the clauses having a literal of v_k remain unchanged in $F(G_k)$ and $F(G_p)$).

C. Construction of tests detecting stuck-at faults

Suppose the stuck-at fault ϕ above occurred in the copy N' of N (i.e. it occurred on the input line v'_k of the AND gate $G_i(v'_k,v'_m)$ 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 $v'_k \vee \sim v'_i$ 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 $v'_k \vee \sim v'_i$ then N'' (the correct copy of N) and N' (the faulty copy) produce different output values. Besides, since $\boldsymbol{p}$ falsifies $v'_k \vee \sim v'_i$ and satisfies the clause $v'_m \vee \sim v'_i$, the assignments to the variables of G'_i are v'_k=0,v'_m=1,v'_i=1. That is the output of G'_i has exactly the (incorrect) value, that would have been produced if v'_k got stuck at 1.

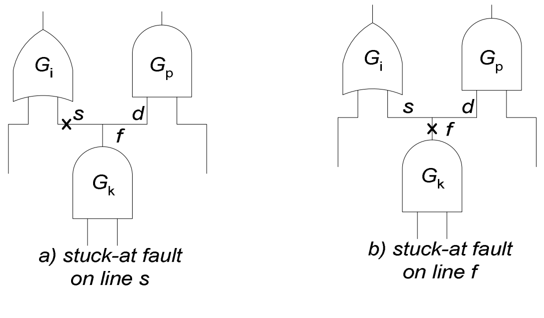

Figure 5. Examples of stuck-at faults

As we mentioned above, a stuck-at-1 fault on line v'_k of G'_i is described by disappearance of clause $v'_k \vee \sim v'_i$ and removing literal $\sim v'_k$ from the clause $C=\sim v'_k \vee \sim v'_m \vee v'_i$. Since any point falsifying $v'_k \vee \sim v'_i$ satisfies the modified clause C (i.e. $\sim v'_m \vee v'_i$), we can “forget” about the disappearance of literal $\sim v'_k$ from C.

If a point $\boldsymbol{p}$ falsifying $v'_k \vee \sim v'_i$ and satisfying the rest of the clauses of F does not exist, then the stuck-at-1 fault on input line v'_m is **untestable** (i.e. this fault does not change the input/output behavior of N).

D. “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 are derived. (Inputs of gates of topological level 1 are inputs of N' and N'') Then, using the equivalence clauses relating outputs of gates of topological level 1, the equivalence clauses relating the outputs of corresponding gates of topological level 2 are generated and so on.

When building the proof R_{nat}, we resolve clauses $F(G'_i(v'_k,v'_m))$ and $F(G''_i(v''_k,v''_m))$ describing corresponding gates G'_i and G''_i of N' and N'' and equivalence clauses $EQ(v'_k,v''_k) \wedge EQ(v'_m,v''_m)$ relating inputs of G'_i and G''_i. (Recall that $EQ(v'_k,v''_k)=(v'_k \vee \sim v''_k) \wedge (\sim v'_k \vee v''_k)$). By resolving clauses of $F(G'_i) \wedge F(G''_i) \wedge EQ(v'_k,v''_k) \wedge EQ(v'_m,v''_m)$ we generate new equivalence clauses $EQ(v'_i,v''_i)$ relating the outputs of G'_i and G''_i.

For example, if G_i of N is an AND gate, then $F(G'_i(v'_k,v'_m))= (\sim v'_k \vee \sim v'_m \vee v'_i) \wedge (v'_k \vee \sim v'_i) \wedge (v'_m \vee \sim v'_i)$ and $F(G''_i(v''_k,v''_m))= (\sim v''_k \vee \sim v''_m \vee v''_i) \wedge (v''_k \vee \sim v''_i) \wedge (v''_m \vee \sim v''_i)$. By resolving $F(G''_i(v''_k,v''_m))$ with clauses of $EQ(v'_k,v''_k)=(v'_k \vee \sim v''_k) \wedge (\sim v'_k \vee v''_k)$ one obtains $F(G''_i(v'_k,v''_m))$ i.e. one replaces variable v''_k with v'_k. (Namely, by resolving $\sim v''_k \vee \sim v''_m \vee v''_i$ with $\sim v'_k \vee v''_k$ the clause $\sim v'_k \vee \sim v''_m \vee v''_i$ is produced. By resolving $v''_k \vee \sim v''_i$ with $v'_k \vee \sim v''_k$ the clause $v'_k \vee \sim v''_i$ is produced.) Similarly, by resolving $F(G''_i(v'_k,v''_m))$ with $EQ(v'_m,v''_m)$ one can obtain formula $F(G''_i(v'_k,v'_m))$ (where variable v''_m is replaced with v'_m.) Note that now formulas $F(G'_i(v'_k,v'_m))$ and $F(G''_i(v'_k,v'_m))$ have the same input variables v'_k,v'_m.

After obtaining $F(G''_i(v'_k,v'_m))$ (and so making input variables of G'_i and G''_i “identical) the two clauses of $EQ(v'_i,v''_i)$ can be obtained in the following way. By resolving the clause $\sim v'_k \vee \sim v'_m \vee v'_i$ of $F(G'_i(v'_k,v'_m))$ with clauses $v'_k \vee \sim v''_i$, $v'_m \vee \sim v''_i$ of $F(G''_i(v'_k,v'_m))$, one generates the clause $v'_i \vee \sim v''_i$ of $EQ(v'_i,v''_i)$. By resolving the clauses $\sim v'_k \vee \sim v'_m \vee v''_i$ of $F(G''_i(v'_k,v'_m))$ with clauses $v'_k \vee \sim v'_i$, $v'_m \vee \sim v'_i$ of $F(G'_i(v'_k,v'_m))$ one obtains the other clause of $EQ(v'_i,v''_i)$ i.e. the clause $\sim v'_i \vee v''_i$.

Eventually, the equivalence clauses relating the corresponding outputs of N' and N'' are derived. By resolving these equivalence clauses with the clauses describing the functionality of the XOR gates and the OR gate of the miter M (see Figure 4) the clause $\sim z$ is derived. By resolving it with the clause z of F (recall that $F=F^*_M \wedge z$) an empty clause is derived.

E. Tight point image of R_{nat} contains tests for all testable stuck-at faults

In this subsection, we give a formal proposition about relation between a circuit test set detecting stuck-at faults and a tight point image of natural resolution proof R_{nat}.

Let N be a combinational circuit consisting of two-input AND or OR gates whose inputs may be negated. Let $F=F^*_M \wedge z$ be the formula describing equivalence checking of copies N' and N'' of N. Let T_{nat} be a tight image of natural proof R_{nat} that F is unsatisfiable. That is for every resolution operation of $Res(C',C'')$, T_{nat} contains points $\boldsymbol{p'}$ and $\boldsymbol{p''}$ that form a point image of $Res(C',C'')$ and falsify the *smallest* possible number of clauses of F. To build T_{nat} one needs to use the *relaxed* point image definition (Section IV.B) where assignments to the variables of $\boldsymbol{p'}$ and $\boldsymbol{p''}$ that are not in $Vars(C') \cup Vars(C'')$ can be made *arbitrarily*.

Usually, by a tight image we mean a “good” approximation in terms of the number of falsified clauses. However, for the proposition below we assume that the variables of $\boldsymbol{p'}$ and $\boldsymbol{p''}$ that are not in $Vars(C') \cup Vars(C'')$ are assigned in an *optimal* way (i.e. so as to *minimize* the number of clauses of F falsified by $\boldsymbol{p'}$ and $\boldsymbol{p''}$**).**

***Proposition* 2.** A set $inp(T_{nat})$ detects all testable stuck-at faults of N .

The **proof** is given in Appendix.

F. Test set $inp(T_{nat})$ is “stronger” than a stuck-at fault test set

The resolution operations of R_{nat} can be partitioned into two sets R^i_{nat} and R^d_{nat}. The resolution operations of R^i_{nat} “identify” inputs of gates G'_i and G''_i by producing $F(G''_i(v'_k,v'_m))$ from $F(G''_i(v''_k,v''_m)) \wedge EQ(v'_k,v''_k) \wedge EQ(v'_m,v''_m)$. The resolution operations of R^d_{nat} derive the equivalence clauses of $EQ(v'_i,v''_i)$ from $F(G'_i(v'_k,v'_m)) \wedge F(G''_i(v'_k,v'_m))$. In its turn R^i_{nat} can be partitioned into R^{i1}_{nat} and R^{i2}_{nat} . Here R^{i1}_{nat} consists of the resolution operations involving equivalence clauses of the *original* formula (i.e. equivalence clauses relating input variables of N' and N''). R^{i2}_{nat} consists of the resolution operations involving *derived* equivalence clauses (i.e. equivalence clauses relating outputs of corresponding gates).

Denote by T^{i1}_{nat} , T^{i2}_{nat}, T^d_{nat} the sets obtained by taking the union of tight point images of resolution operations from R^{i1}_{nat}, R^{i2}_{nat}, R^d_{nat} respectively. Then a point image T_{nat} of R_{nat} can be represented as $T^{i1}_{nat} \cup T^{i2}_{nat} \cup T^d_{nat}$. In the proof of Proposition 2, we used only resolution operations of R^{i1}_{nat} and R^d_{nat}. This means that the circuit test set inp(T^{i1}_{nat}) $\cup$inp(T^d_{nat}) contains tests detecting all testable stuck-at faults and hence inp(T_{nat}) is a “stronger” test set.

Figure 6 helps explain in what sense $inp(T_{nat})$ is a stronger test set. Let $G'_i(v'_k,v'_m)$ and $G''_i(v''_k,v''_m)$ be corresponding gates of N' and N''. When generating the formula $F(G''_i(v'_k,v'_m))$ from $F(G''_i(v''_k,v''_m))$, one needs to resolve, say, clause $C_1 = v''_k \vee \sim v''_i$ of $F(G''_i(v''_k, v''_m))$ with clause $C_2 = v'_k \vee \sim v''_k$ of $EQ(v'_k, v''_k)$. Let $\boldsymbol{p}$ be the point of a tight image of $Res(C_1, C_2)$ that falsifies C_2. Then $v'_k = 0$ and $v''_k = 1$. This means that $\boldsymbol{p}$ has to falsify a clause of a gate G''_f of $N''(G''_i)$. Disappearance of this clause corresponds to a stuck-at fault ϕ on a input/output line of G''_f (we assume here that N'' is the faulty circuit).

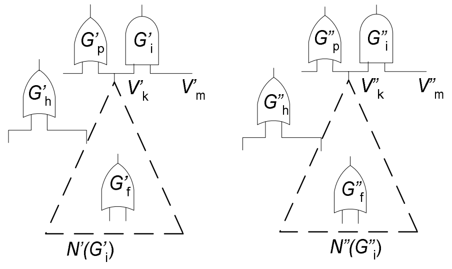

Figure 6. Lack of fault propagation through gates G'_i, G''_i

Note that even though $inp(T^{i1}_{nat} \cup T^{d}_{nat})$ detects all testable stuck at faults (and hence the stuck-at fault ϕ), the set $T^{i1}_{nat} \cup T^{d}_{nat}$ may not contain such a point $\boldsymbol{p}$. Namely it may not contain a point falsifying the clause C_2 and the resolvent $C_3 = v'_k \vee \sim v''_i$ of C_1, C_2 and. Indeed, suppose that for every point $\boldsymbol{p}^*$ of $T^{i1}_{nat} \cup T^{d}_{nat}$ falsifying C_2, the value of v''_m is 0. Then $v''_i = 0$ too and hence the resolvent C_3 is satisfied by $\boldsymbol{p}^*$. Note that the assignments $v'_m = 0, v''_m = 0$ are "non-propagating assignments of G'_i and G''_i. That is even if v'_k and v''_k have different values caused by the fault ϕ in G''_f, the outputs of G'_i and G''_i will be 0 if $v'_m = 0, v''_m = 0$. (However, in general, we cannot claim that if $v''_m = 0$, then $v'_m = 0$ too because the value of v''_m may be affected by the fault ϕ)

The situation above means that even though $inp(T^{i1}_{nat} \cup T^{d}_{nat})$ detects all stuck-at faults in subcircuit $N''(G''_i)$, none of these faults propagates through the gate G''_i. (They may propagate through a gate G''_p fed by the same gate G''_k as G''_i or through some gate G''_h fed by an internal gate of $N''(G''_i)$.) Summarizing, one can say that the difference between $inp(T^{i1}_{nat} \cup T^{d}_{nat})$ (and hence between a "regular" test set detecting all testable stuck-at faults) and $inp(T_{nat})$ is as follows. The circuit test set $inp(T_{inp})$ not only detects all testable stuck-at faults, but also "diversifies" their propagation. (In particular, it will try to get a test $inp(\boldsymbol{p}_i)$ that detects a fault in a gate G''_f of $N''(G''_i)$ and "propagates" the faulty value through the gate G''_i.)

G. *A few concluding remarks*

The size of the R_{nat} above (and hence a tight point image T_{nat} of R_{nat}) is linear in the size of N. Moreover, since different points of T_{nat} may have identical input parts, the size of the circuit test set produced from T_{nat} may be considerably smaller than that of T_{nat}. Importantly, T_{nat} is not built 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.

As we showed above, $inp(T_{nat})$ is "stronger" than a regular test set detecting stuck-at faults. One can generate an *even stronger* test set by "rarifying" the proof R_{nat}. 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_{nat} that were generated from clauses $F_{K'}$ and $F_{K''}$ when obtaining 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 $EQ(v'_i, v''_i)$. An SSP S of the subformula F_C (obtained from F by making the assignments falsifying the literals of C) will contain more points than the corresponding part of a point image of R_{nat}. (We mean the part that involves clauses of $F_{K'}$ and $F_{K''}$). So a test set T containing S will provide more "thorough" testing of the subcircuit K of N.

VII. 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. Below we show 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 N. 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.

A. *Testing a modified circuit*

Suppose that we managed to generate a resolution proof R that F is unsatisfiable (and so 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'. Note that generating a *tight* test set may be time consuming. So building a tight sufficient test set makes more sense when the number of changes in N is large. (Another solution is to build a "partial" tight point image. Namely one can pick a subset R^* of resolution operations from R and build T as the union of tight point images of the resolution operations from R^* . The experimental results given in Subsection VIII.C imply that circuit tests extracted even from a partial image have high quality.

B. *Testing property 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 we try to reduce to 0 the number of original clauses of F falsified by $\boldsymbol{p}$ at the expense of falsifying *any number* of clauses of H. 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 of R 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 following sense. When "attacking" (i.e. falsifying) assumptions of H we take into account every situation in which an assumption clause (or its descendent in R) was used in R.

VIII. Experimental Results

In this section, we give some experimental results. In Subsection A we show that checking the sufficiency of a test set without lemma clauses is inefficient. Subsection B gives some data indicating that sufficient tests are capable of detecting small variations of random CNF formulas. In Subsection C, we compare circuits tests generated randomly and ones extracted from a tight point image of a resolution proof.

In all experiments of Subsections A and B (except the one described in Table 4) we used *"irredundant" random formulas*. Each formula was obtained by generating a random unsatisfiable CNF formula from the hard domain [7] and then removing all redundant clauses from this formula. (A clause is *redundant* in an unsatisfiable formula if removing it from this formula does not make the latter satisfiable.)

A. Recovering resolution proof without lemma clauses

The objective of the experiment described in this subsection was to show that checking if a test set is sufficient can not be done *efficiently* without lemma clauses.

In Table 1, we use the procedure sketched in [2] for recovering a resolution proof from a point image. The procedure of [2] is essentially meant to check if a set of points T is a sufficient test set for a CNF formula F. The idea of this procedure is to resolve every pair of clauses C',C'' of the current formula *only* if T has a point image of this resolution operation. That is proof generation is guided by T *without* using any lemma clauses.

name	#vars	#clauses	image (size)	proof (size)	recov. #res. op.
F_1	10	16	44	19	129
F_2	15	29	119	56	1867
F_3	20	39	132	62	3258
F_4	25	58	472	245	> 100000

Table 1. Recovering proofs *without* lemma clauses

In the experiment, we first generated a resolution proof R that a formula F_n is unsatisfiable and built a point image T of R. Then we used an implementation of the procedure of [2] to "recover" the proof R. The second and third columns give the number of variables and clauses of the formulas. The fourth column give the size of the point image that was extracted from the proof . The proof size (in the number of resolution operations) is shown in fifth column. The last column of Table 1 shows the number of resolution operations performed before generating an empty clause.

At first glance, the choice of clauses to resolve is very restricted in [2]. Besides, we took all possible measures to minimize the number of "useless" resolution operations. In particular, the test set T was made tight. Besides, a resolvent C was not added to the current formula if the latter had a clause implying C. Nevertheless, experimental results show that these restrictions are insufficient and the number of "junk" resolution operations grows exponentially. This problem is solved in $Sat(T,F,L)$ and $Sat^*(T,F,L,A)$ procedures described in our report. We do not give experimental results for them because these procedures are *provably* efficient and by definition generate only "useful" clauses i.e. lemmas. (So generation of junk resolvents is not possible.)

B. Using sufficient test sets for fault detection in random CNF formulas

In this subsection, we describe application of sufficient test sets for detection of small variations (further referred to as faults) of unsatisfiable random formulas. In all experiments, we considered the same type of a "single" fault, namely adding a literal to a clause of the formula. We used only "testable" faults i.e. ones that made the formula satisfiable. In all experiments, for every formula we generated 100 single faults (that is we produced 100 satisfiable formulas) and checked how many of them were detected by random and sufficient test sets. (Both random and sufficient test sets were applied to the *same* sets of faults.). In all experiments, we first built a sufficient test set and then generated a random test set of the *same size*.

In all experiments, except the one described in Table 5, we generated just a sufficient test set. Building a *tight* sufficient is beneficial when the subformula F_{L_i} (obtained from F by making the assignments falsifying all the literals of the lemma clause L_i) has a very small unsatisfiable subset of clauses F'_{L_i}. Then one can assign the variables of $Vars(F'_{L_i}) \setminus Vars(F)$. arbitrarily (assuming that we relax the definition of a sufficient test set as described in Subsection IV.A). However, in experiments described in Table 2 and Table 4 we built SSPs (set L consisted only of an empty clause). In the experiment described in Table 3 we built a test set with the corresponding set L of two unit clauses and one empty clause (that is very close to an SSP).

In Table 2, we compare SSPs and random test sets. The second column of the table gives the number of variables. The third column shows the number of clauses in the irredundant CNF formula while the number in the parenthesis gives the number of clauses in the original (redundant) formula. The number of points in the obtained SSP is given in the fourth column. (For generating SSPs we used the procedure described in [3] and[4].) The number of faults (out of 100) detected by an SSP and a random test set of the same size is shown in the last two columns. The main conclusion one can draw from Table 2 is that SSPs and random tests are competitive in fault detection for the set of irredundant random formulas we used.

In the next experiment (Table 3) we used the same CNF formulas as in Table 2. To generate a sufficient test set of a formula G_i we picked a variable x of G_i, built SSPs S_0 and S_1

Name	#vars	#clauses (#orig.)	SSP size	#faults (rand)	#faults (SSP)
G_1	19	34 (80)	69,836	98	98
G_2	19	34 (80)	118,350	100	100
G_3	20	41(85)	147,619	91	96
G_4	20	41(85)	121,071	92	94
G_5	21	37(89)	142,018	99	93
G_6	21	45(89)	407,523	88	100

Table 2. Testing *irredundant* formulas randomly and by SSP

for subformulas $G_i(x=0)$ and $G_i(x=1)$ and took the union of S_0 and S_1. (The fact that the set $T=S_0 \cup S_1$ is sufficient for G_i can be established by the procedure $Sat(T,G_i,L)$ where L consists of lemma clauses x,~x and an empty clause.) The size of S_0,S_1 and $S_0 \cup S_1$ is given in second, third and fourth columns respectively. In this experiment, we compared the performance of test set $S_0 \cup S_1$ with two kinds of random test sets. The first random test set (its results are shown in the column 'rnd$_1$') was obtained just by generating randomly a set of tests of the same size as $S_0 \cup S_1$. The second random test set was obtained by generating $|S_0|$ tests for subformula $G_i(x=0)$ (i.e. for all tests x=0) and $|S_1|$ tests for subformula $G_i(x=1)$ (i.e. for all tests x=1) and then taking the union of these two sets.

One can draw two conclusions from Table 3. First, although test sets $S_0 \cup S_1$ are 2-4 times smaller than the corresponding SSPs (Table 2) their performance (in terms of detected faults) remains almost the same. Second, the

Name	SSP$_0$ (size)	SSP$_1$ (size)	SSP$_0 \cup$ SSP$_1$	#flts rnd$_1$	#flts rnd$_2$	#flts SSP$_0$ $\cup$ SSP$_1$
G_1	12,059	19,921	31,980	77	95	98
G_2	9,299	23,012	32,311	93	82	98
G_3	11,906	22,464	34,370	73	73	95
G_4	16,443	29,846	46,289	72	70	94
G_5	24,016	12,500	36,516	82	77	91
G_6	87,635	59,420	147,055	75	80	100

Table 3. Testing formula by decomposing it into subformulas

performance of random test sets of the same size drops significantly. This drop may be attributed to the fact that in contrast to formula G_i that is irredundant, subformulas $G_i(x=0)$ and $G_i(x=1)$ are, in general, redundant. So one could explain the results of Table 3 by performance degradation of random test sets for redundant formulas

The intuition above is confirmed in our next experiment (Table 4). In this experiment, we used the same *initial* CNF formulas as in the previous experiments. The difference was that only a part of the redundant clauses was removed. More precisely, G^*_i contains the same clauses as G_i plus 20 redundant clauses of the original random formula. We built an SSP of G^*_i and compared its performance with a random test set of the same size. An obvious conclusion is that the performance of random test sets drops significantly. (One should not read much into the fact that SSPs better performed for redundant formulas than for irredundant ones detecting all 100 faults in each case. SSPs of formulas G^*_i are larger than those of formulas G_i.) Note that no matter "how much" a CNF formula is irredundant its subformulas are redundant. This is especially true for formulas representing circuits. It is not uncommon that a value assignment to a circuit variable makes a very large part of this circuit unobservable and hence redundant.

Name	#clauses (#orig.)	SSP size	#faults (rand)	#faults (SSP)
G^*_1	54(80)	116,206	84	100
G^*_2	54(80)	110,450	75	100
G^*_3	61(85)	205,378	61	100
G^*_4	61(85)	177,881	47	100
G^*_5	57(89)	416,395	81	100
G^*_6	65(89)	348,849	69	100

Table 4. Testing *redundant* formulas randomly and by SSP

In the final experiment (Table 5), we compared the performance of a *tight* point image of a resolution proof with a random test set of the same size. Even though a point image is the "weakest" kind of a sufficient test set, it managed to detect a significant number of faults. On the other hand, the performance of a random test set of the same size dropped dramatically.

Name	Proof size	Image size	#faults (rand)	#faults (image)
G_1	63	58	0	82
G_2	48	42	5	64
G_3	95	76	0	67
G_4	93	75	7	72
G_5	82	64	0	80
G_6	121	87	0	68

Table 5. Testing formulas randomly and by a *tight* point image of a resolution proof

A tight point image T of a proof R that a CNF F is unsatisfiable was generated as follows. Let C be a resolvent of R obtained from the parent clauses C' and C'' by resolving them in variable x_i. Let C^0 and C^1 be the clauses obtained from C by adding the negative and positive literals of x_i respectively. Any pair of points $\boldsymbol{p'}$ and $\boldsymbol{p''}$ falsifying C^0 and C^1 form a point image of the resolution operation over C' and C''. (We assume here that C' and C'' contain the positive and negative literals of x_i respectively) To make, this image tight one needs to find points p' and p'' that falsify as few clauses of the initial formula as possible. To find this tight image we generated subformulas F_{C0} and F_{C1} and looked for complete assignments falsifying as few clauses of F_{C0} and F_{C1} as possible. (Here F_{C0}, F_{C1} are the CNF formulas obtained from the initial formula F by setting to 0 all the literals of C^0 and C^1 respectively.)

Initially the set T was empty. Then for every resolution operation over clauses C',C'' we looked for points falsifying F_{C0} and F_{C1} and added them to T. When generating, for example, point p' we used Walksat [6] to find a complete assignment falsifying as few clauses of F_{C0} as possible. If a point falsifying F_{C0} was already in T, no new point was generated.

The resolution proofs in this experiment (and the experiment of the following subsection were generated by the SAT-solver *FI* [2].

C. *Using sufficient test sets for fault detection in circuit CNF formulas*

In this subsection, we describe application of tight sufficient test sets to circuit testing. In the experiments, we compared the quality of *circuit* tests (i.e. assignments to input variables) generated randomly and extracted from a tight sufficient test set. Given a circuit N, a tight sufficient test set T was extracted from a resolution proof R that a formula F describing equivalence checking of two copies of N is unsatisfiable. (T was extracted from R as described in end of the previous subsection). A circuit test set was derived from a point image T as follows. First, we formed the set $inp(T)$ consisting of the input parts of the points of T, then we *randomly* extracted a specified number of circuits tests from $inp(T)$.

The circuits we used are shown in Table 6. All the circuits are taken from a standard MCNC benchmark set. The original circuits were transformed by the logic synthesis system SIS [8] to ones consisting of two-input AND and OR gates inputs of which may be negated.

The second, third and fourth columns give the number of inputs, outputs and gates of the circuit. The fifth column

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
c1908	33	25	635	49,642	26,425
cm150a	21	1	77	881	603
comp	32	3	146	1,797	1,374
count	35	16	143	2,349	1,831
frg1	28	3	792	42,700	20,351
i2	201	1	233	1,777	1,435
mux	21	1	136	1,777	1,181
term1	34	10	854	83,475	43,718

Table 6. The size of circuits, proofs and point images

shows the size of the proof R found by *FI* [2] (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).

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'' (formula F^*_M was introduced in Subsection VI.A). We generated faults of the same type as in the previous subsection. That is a fault is to add a literal to a clause of F^*_M. Let $\boldsymbol{s}$ be a circuit test (i.e. an assignment to the input variables of N). To check if a fault ϕ above is detected by $\boldsymbol{s}$ we make the assignments specified by $\boldsymbol{s}$ in F and run Boolean Constraint Propagation (BCP) over 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 in F^*_M 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 $C=\sim v'_i \vee \sim v'_j \vee v'_k$ be a clause of the CNF $F(G'_k)$ describing the functionality of the AND gate $G'_k(v'_i,v'_j)$ of N'. Suppose that ϕ is to add the literal v'_m to C. In the correct gate $G'_k(v'_i,v'_j)$, if $v'_i=1,v'_j=1$, then the value $v'_k=1$ is derived from the clause C. However, if the value of v'_m is equal to 1 under input assignment $\boldsymbol{s}$, then the clause $\sim v'_i \vee \sim v'_j \vee v'_k \vee v'_m$ is satisfied even without setting the value of v'_k to 1. (Another possibility is that v'_m is still unassigned by the time assignments $v'_i=1,v'_j=1$ are made during BCP. Then one can not deduce the assignment $v'_k = 1$ from the clause $v'_k \vee v'_m$.) So the output of the gate G'_k remains unspecified under the input assignment $\boldsymbol{s}$.

In the case z remains unassigned in F^*_M after performing BCP, we run a SAT-solver to see if one can assign the rest of the variables to satisfy F (and so set z to 1). In other words, we consider a input assignment **s** as detecting a literal appearance fault, if one can extend $\boldsymbol{s}$ to a satisfying assignment. Otherwise, $\boldsymbol{s}$ does not detect this fault.

The literal appearance fault does not exactly correspond to an existing model of a manufacturing fault. On the one hand, it is not very important, because the existing fault models give only a rough approximation of real technological faults. On other hand, literal appearance does have a *"technological interpretation"*. Suppose that the fault ϕ above occurred in gate $G'_k(v'_i,v'_j)$. Adding literal v'_m to the clause $C=\sim v'_i \vee \sim v'_j \vee v'_k$ describes (unwanted) interaction between gates G'_k and G'_m. (Such kind of interaction is caused by high density of electrical components of a chip). Namely, the value taken by the output of G'_k (under input assignments $v'_i = 1$, $v'_j = 1$) becomes unpredictable if G'_m evaluates to 1. As it was mentioned above, we consider an input assignment $\boldsymbol{s}$ as detecting ϕ, if $\boldsymbol{s}$ can be "extended" to an assignment satisfying F. This means that we consider $\boldsymbol{s}$ as detecting ϕ, if $\boldsymbol{s}$ detects this fault , when G'_k "chooses" to produce the wrong value of its output (i.e. 0).

Importantly, a literal appearance fault is more "subtle" than a stuck-at fault that can be simulated by removing a clause. Besides, tests detecting literal appearance in a clause of F^*_M can be also used simulate small design changes that are "hard to detect" in functional verification.

Table 7 shows the results of fault testing for the circuits of Table 6. In every experiment we generated 100 testable faults (i.e. every fault made *F* satisfiable). The second column of Table 7 gives the size of a test set. The third column gives the results of fault detection using a test set detecting all stuck-at faults in *N*. This test set was obtained by SIS [8]. 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. (This test set finds all testable stuck-at faults in this circuit.) These tests were able to detect 86 out of 100 testable faults of literal appearance.

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)
c1908	137	86	65.7(59)	81.8 (76)
	300	-	72.5 (68)	87.3 (85)
	600	-	82.8 (79)	93.1(90)
cm150a	40	75	38.9 (26)	55.6 (46)
	100	-	62.8 (53)	83.2 (80)
	200	-	78.9 (65)	95.6 (89)
comp	86	91	29.9 (26)	82.5 (75)
	200	-	37.0 (32)	92.5 (87)
	400	-	45.2 (39)	97.1 (96)
count	35	86	74.6 (68)	89.6 (85)
	70	-	85.5 (83)	96.2 (94)
	140	-	88.2 (85)	97.8 (96)
frg1	172	87	27.5(24)	55.8 (51)
	300	-	32.5(29)	68.4 (66)
	600	-	38.7(35)	77.7 (72)
i2	221	71	7.8(3)	66.4(62)
	400	-	9.2(6)	74.6(69)
	600	-	11.6(10)	82.4(80)
mux	44	75	37.8(27)	58.7 (48)
	100	-	51.7 (43)	83.3 (79)
	200	-	67.8 (62)	91.2 (88)
term1	323	79	35.0 (28)	63.7 (58)
	600		44.6 (40)	76.4 (72)
	1200		58.8 (55)	87.0 (84)

Table 7. Circuit testing

The fourth column contains the results of fault detection when using circuit tests generated randomly. In every experiment, we generated 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 7) 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 results of fault detection when using circuit tests extracted from the set *inp*(*T*) where *T* is a point image of a proof *R* that $F^{*}_{M} \wedge z$ is unsatisfiable. Namely we randomly extracted a particular number of tests from *inp*(*T*). The corresponding sizes of *T* are given in Table 6. 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, for the circuit *c432*, 10 test sets of 58 circuit tests each were extracted from *inp*(*T*). The average number of detected faults was 79.7 and the worst result was 76 detected faults.

One can draw from Table 7 the following three conclusions. 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 a test set detecting the stuck-at faults have high-quality, it does not detect all literal appearance faults. Third, tests extracted from a tight point image *T* of a resolution proof *R* perform better than random tests. For circuits like *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 like *cordic* and *i2* that are also shallow but have only 2 and 1 outputs respectively, tests extracted from resolution proofs significantly outperformed random tests.

IX. Conclusion

In this report, we develop a theory of sufficient test sets. The essence of our approach is to use a set of points as a “prover” and measure the “completeness” of a test set by its “proving power”. 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 (possibly “rarified”). 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. *Determinization of resolution by an algorithm operating on complete assignments.* SAT-2006, LNCS 4121, pp.90-95.

[3] E. Goldberg. *Testing Satisfiability of CNF Formulas by Computing a Stable Set of Points*, CADE 2002, LNCS, vol 2392,pp.161-180.

[4] E.Goldberg. *Testing Satisfiability of CNF Formulas by computing a Stable Set of Points.* Annals of Mathematics and Artificial Intelligence, 43 (1-4):65-89, January 2005.

[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] B. Selman, D.Mitchell, H.Levesque . *Generating Hard Satisfiability Problems*. Artificial Intelligence, Vol. 81, 1996, 17--29.

[8] E.M. Sentovich et. al. *SIS: A system for sequential circuit synthesis*. Technical report, University of California at Berkeley, 1992. Memorandum No. UCB/ERL M92/41

Appendix.

***Proof of Proposition* 2.** Below we consider stuck-at faults occurring either in a gate G'_i of N' or in the corresponding gate G''_i of N''. Since N' and N'' are identical copes of N it does not matter which one currently serves as a faulty copy.

We consider only the case of a two-input AND gate whose inputs are not negated. The case of OR and AND gates where an input (or both inputs) are negated can be proven similarly. We assume that all faults we consider in the proof are testable. If a fault is untestable, then $inp(T_{nat})$ does not have to contain a test for this fault (and it does not contain such a test). In the proof below, we look for relaxed point images of resolution operations (introduced in IV.B). We partition the proof into three steps that consider the stuck-at faults on input lines of gates of N, stuck-at faults on output lines of gates of N and stuck-at faults on primary input lines of N.

Stuck-at faults on input lines of gates.

Let $G'_i(v'_k,v'_m)$ and $G''_i(v''_k,v''_m)$ be two corresponding AND gates of N' and N''. After resolving $F(G''_i(v''_k,v''_m))$ with clauses of $EQ(v'_k,v''_k) \wedge EQ(v'_k,v''_k)$ we produce the formula $F(G''_i(v'_k,v'_m))$. Let us consider resolution operations over clauses of $F(G'_i(v'_k,v'_m))= (\sim v'_k \vee \sim v'_m \vee v'_i) \wedge (v'_k \vee \sim v'_i) \wedge (v'_m \vee \sim v'_i)$ and $F(G''_i(v'_k,v'_m))= (\sim v'_k \vee \sim v'_m \vee v''_i) \wedge (v'_k \vee \sim v''_i) \wedge (v'_m \vee \sim v''_i)$. Denote by C_1 the clause $\sim v'_k \vee \sim v'_m \vee v'_i$ of $F(G'_i(v'_k,v'_m))$ and by C_2,C_3 the clauses $v'_k \vee \sim v''_i$, $v'_m \vee \sim v''_i$ of $F(G''_i)$ respectively. For the sake of clarity, we assume that the clause $v'_i \vee \sim v''_i$ of $EQ(v'_i,v''_i)$ is obtained by operations $Res(C_1,C_2)$ and $Res(C_4,C_3)$ where $C_4{=}\sim v'_m \vee v'_i \vee \sim v''_i$ is the resolvent of C_1,C_2. (Resolving C_1 first with C_3 and then with C_2 can considered similarly.)

This case is partitioned into the following three subcases: 1) Stuck-at-0 faults on input lines v'_k,v'_m of G'_i. 2) Stuck-at-1 fault on input line v''_k of G''_i; 3) Stuck-at-fault-1 on input line v''_m of G''_i.

1) *Stuck-at-0 fault on input lines v_k,v_m of G'_i.*
Since the stuck-at-0 fault on input line v'_k is testable, one can pick a point $\boldsymbol{p}$ that falsifies the clause $C_1{=}\sim v'_k \vee \sim v'_m \vee v'_i$ and satisfies all other clauses of F. Note that any such point $\boldsymbol{p}$ falsifies the resolvent $\sim v'_m \vee v'_i \vee \sim v''_i$ of C_1 and $C_2 = v'_k \vee \sim v''_i$. Indeed, since $\boldsymbol{p}$ falsifies C_1, then $v'_m{=}1,v'_i{=}0$. Since $\boldsymbol{p}$ satisfies the clause $C^*{=} \sim v'_k \vee \sim v'_m \vee v''_i$ of $F(G''_i(v'_k,v'_m))$, then $v''_i{=}1$.(Note that C^* is a *derived* clause. So one can not claim that $\boldsymbol{p}$ satisfies C^* just because it satisfies all the clauses of the *original* formula but C_1. The reason why $\boldsymbol{p}$ satisfies C^* is that it satisfies the subformula $F(G''_i(v''_k,v''_m)) \wedge EQ(v'_k,v''_k) \wedge EQ(v'_m,v''_m)$ from which C^* was derived. $F(G''_i(v''_k,v''_m))$ is satisfied by $\boldsymbol{p}$ because it consists of clauses of the original formula. The clauses $EQ(v'_k,v''_k) \wedge EQ(v'_k,v''_k)$ are satisfied because they are implied by the conjunction of the clauses $F(G'_p)$, $F(G''_p)$ of all the gates G'_p, G''_p located below the gates G'_i and G''_i.) So there is a point $\boldsymbol{p_1}$ falsifying C_1 that satisfies all the other clauses of F and any such point satisfies the resolvent of C_1,C_2 This essentially means that any point $\boldsymbol{p_1}$ of a *tight* image of $Res(C_1,C_2)$ falsifying C_1, satisfies the other clauses of F. Then the circuit test $inp(\boldsymbol{p_1})$ detects the stuck-at 0 fault on input line v_k of G_i of N.

Test $inp(\boldsymbol{p}_1)$ also detects the stuck-at-0 fault on input line v_m of G_i of N. Indeed, suppose that $inp(\boldsymbol{p_1})$ is applied to the inputs of the faulty circuit N'. Then the outputs of the gates G'_k and G'_m described by variables v'_k, v'_m evaluate to 1. If there is a stuck-at-0 fault on the input line v'_k of G'_i, the rest of the gates fed by G'_k (and all the gates fed by G'_m) still obtain input value 1. If there is a stuck-at-0 fault on the input line v'_m of G'_i, the rest of the gates fed by G'_m (and all the gates fed by G'_k) still obtain value 1. That is under stuck-at-0 faults on lines v'_k and v'_m, assignments to the rest of the variables of the faulty circuit N' (including the output of G'_i which is equal to 0) are the same.

2) *Stuck-at-1 fault on input line v''_k of G''_i.*

Since the stuck-at-1 fault on input line v''_k is testable, one can build a point $\boldsymbol{p}$ that falsifies the clause $v''_k \vee \sim v''_i$ and satisfies the rest of the clauses of F. Note that since $\boldsymbol{p}$ satisfies the clauses of $EQ(v'_k,v''_k)$ (for the reason explained above), it falsifies the clause $C_2{=} v'_k \vee \sim v''_i$ of $F(G''_i(v'_k,v'_m))$. Besides $\boldsymbol{p}$ falsifies the resolvent $C_4{=}\sim v'_m \vee v'_i \vee \sim v''_i$ of C_1 and C_2. (Indeed, since $\boldsymbol{p}$ falsifies C_2, then $v'_k{=}0,v''_i{=}1$. Since $\boldsymbol{p}$ satisfies the clause $v'_k \vee \sim v'_i$ of $F(G'_i)$, then $v'_i{=}0$. Since $\boldsymbol{p}$ satisfies the clause $v'_m \vee \sim v''_i$ of $F(G'_i)$, then $v'_m{=}1$.). This means that any point $\boldsymbol{p_2}$ of a *tight* image of $Res(C_1,C_2)$ falsifying C_2, satisfies the other clauses of F. (This follows from the fact that any clause falsifying C_2, falsifies $v''_k \vee \sim v''_i$ as well.) Then the circuit test $inp(\boldsymbol{p_2})$ detects the stuck-at-1 fault on input line v_k of G_i of N.

3) *Stuck-at-fault-1 on line v''_m of G''_i.* Since the stuck-at-1 fault on input line v_m is testable, one can build a point $\boldsymbol{p}$ that falsifies the clause $v''_m \vee \sim v''_i$ of $F(G''_i(v''_k,v''_m))$ and satisfies the rest of the clauses of F. Note that any such a point $\boldsymbol{p}$ also satisfies the clauses of $EQ(v'_m,v''_m)$ and hence falsifies the clause $C_3{=}v'_m \vee \sim v''_i$ of $F(G''_i(v'_k,v'_m))$. Besides, $\boldsymbol{p}$ falsifies the resolvent $v'_i \vee \sim v''_i$ of $C_4{=}\sim v'_m \vee v'_i \vee \sim v''_i$ and C_3. (Indeed, since $\boldsymbol{p}$ falsifies C_3, then $v'_m{=}0,v''_i{=}1$. Since $\boldsymbol{p}$ satisfies the clause $v'_m \vee \sim v'_i$ of $F(G'_i)$, then $v'_i{=}0$). This means that any point $\boldsymbol{p_3}$ of a *tight* image of $Res(C_4,C_3)$ falsifying C_3, satisfies the other clauses of F. Then the circuit test $inp(\boldsymbol{p_3})$ detects the stuck-at-1 fault on input line v_m of G_i of N.

Stuck-at faults on output lines of gates.

This case consists of the following two subcases.

The stuck-at-0 fault on the output line v'_i of $G'_i(v'_k,v'_m)$. This fault is detected by the circuit test $inp(\boldsymbol{p_1})$ of a tight image of $Res(C_1,C_2)$ that we considered above. (That is $inp(\boldsymbol{p_1})$ is the same circuit test that detects stuck-at 0 on input lines v'_m and v'_k of G').

The stuck-at-1 fault on the output line v''_i of $G''_i(v''_k,v''_m)$. Denote this fault by ϕ. Let us consider the following 3 situations 1) the stuck-at-1 fault ϕ_1 on the input line v''_k of $G''_i(v''_k,v''_m)$ is testable; 2) ϕ_1 is untestable but stuck-at fault ϕ_2 on the input line v''_m of $G''_i(v''_k,v''_m)$ is testable; 3) both ϕ_1 and ϕ_2 are untestable.

1) Suppose that ϕ_1 is testable. Let $\boldsymbol{p_2}$ be the point introduced above. That is $\boldsymbol{p_2}$ is the point a tight point image of $Res(C_1,C_2)$ that falsifies C_2 where $C_1= \sim v'_k \vee \sim v'_m \vee v'_i$ and $C_2=v'_k \vee \sim v''_i$. Then $v'_k=0, v'_m=1, v'_i=0$ and $v''_k=0, v''_m=1, v''_i=1$. That is $inp(\boldsymbol{p_2})$ detects not only ϕ_1 but also ϕ.

2) Suppose that ϕ_1 is untestable but ϕ_2 is testable. Let $\boldsymbol{p_3}$ be the point introduced above. That is $\boldsymbol{p_3}$ is the point of a tight point image of $Res(C_3,C_4)$ that falsifies C_3 where $C_4=\sim v'_m \vee v'_i \vee \sim v''_i$ and $C_3=v'_m \vee \sim v''_i$. Then $v'_k=1, v'_m=0, v'_i=0$ and $v''_k=0, v''_m=1, v''_i=1$. That is $inp(\boldsymbol{p_3})$ detects not only ϕ_2 but also ϕ.

3*) Suppose both ϕ_1 and ϕ_2 are untestable but ϕ is testable. Then there is a point $\boldsymbol{p}$ that falsifies the clauses $C_2=v''_k \vee \sim v''_i$, $C_3=v''_m \vee \sim v''_i$ of $F(G''_i(v''_k,v''_m))$ and satisfies the other clauses of F ($\boldsymbol{p}$ cannot falsify only C_2 or C_3 because neither ϕ_1 nor ϕ_2 are untestable) . Since $\boldsymbol{p}$ satisfies the clauses of $EQ(v'_k,v''_k) \wedge EQ(v'_m,v''_m)$, then it falsifies the clauses $C_2=v'_k \vee \sim v''_i$, and $C_3= v'_m \vee \sim v''_i$. Let $\boldsymbol{p_4}$ be the point of a tight point image of the resolution operation over C_3,C_4 that falsifies C_3. (Recall that $C_4=\sim v'_m \vee v'_i \vee \sim v''_i$ is the resolvent of C_1 and C_2.) Then if one takes into account Remark 1 below, point $\boldsymbol{p_4}$ falsifies *only* clauses C_2 and C_3 of F. So $v'_k=0$, $v'_m=0, v'_i=0$ and $v''_k=0$, $v''_m=0, v''_i=1$. That is $inp(\boldsymbol{p_4})$ detects ϕ.

Stuck-at faults on primary input lines of N'. Let ϕ be a stuck-at-1 fault on input line v'_k of N'. (Note that this fault is different from a stuck-at-1 fault, say, on the input line v'_k of gate $G'_i(v'_k,v'_m)$. The fault ϕ means that the value on input lines v'_k of *all* gates of N' fed by the input line v'_k of N' is stuck at 1.) If ϕ is testable, then there is an assignment $\boldsymbol{p}$ falsifying the clause $C_5=\sim v'_k \vee v''_k$ of $EQ(v'_k,v''_k)$ and satisfying the rest of the clauses of F. (Note that the clauses of $EQ(v'_k,v''_k)$ relating input variables v'_k,v''_k of N' and N'' are a part of the original formula F). Since $\boldsymbol{p}$ falsifies C_5 then $v'_k=1$ and $v''_k=0$. For $\boldsymbol{p}$ to satisfy F there has to be gates $G'_i(v'_k,v'_m)$ and $G''_i(v''_k,v''_m)$ of N' and N'' "propagating the difference between the faulty ($v'_k=1$) and correct ($v''_k=0$) values to a pair of corresponding outputs of N' and N''. (Otherwise, the output value of M will be 0.) For AND gates $G'_i(v'_k,v'_m)$ and $G''_i(v''_k,v''_m)$, it means that $v'_m=v''_m=1$, $v'_i=1, v''_i=0$.

In R_{nat}, one resolves $F(G''_i(v''_k,v''_m))$ with clauses of $EQ(v'_k,v''_k) \wedge EQ(v'_m,v''_m)$ to produce the formula $F(G''_i(v'_k,v'_m))$. Denote by C_6 the clause $\sim v''_k \vee \sim v''_m \vee v''_i$ of $F(G''_i(v''_k,v''_m))$. Since G'_i and G''_i are "propagating" gates then the point $\boldsymbol{p}$ above falsifies the resolvent $\sim v'_k \vee \sim v''_m \vee v''_i$ of C_5,C_6. Then the point $\boldsymbol{p_5}$ of any tight point image of $Res(C_5,C_6)$ that falsifies C_5, satisfies all the other clauses of F. (Here C_6 is the clause $\sim v''_k \vee \sim v''_m \vee v''_i$ of $F(G''_i(v''_k,v''_m))$ describing a fault "propagating" gate G''_i.) Then $inp(\boldsymbol{p_5})$ is a circuit test detecting ϕ.

A stuck-at-0 fault on input line v'_k of N' can be considered in a similar manner

REMARK 1. *Correction of tight image definition.* To make the case 3*) above work one needs to correct the definition of a tight image of resolution operation as follows. Suppose $\boldsymbol{p}, \boldsymbol{p}^*$ are candidates for a point of a tight image of $Res(C_i,C_k)$ falsifying, say, the clause C_i. Suppose $\boldsymbol{p}, \boldsymbol{p}^*$ both falsify two clauses of the initial formula (that is C_i and some other clause) and there is no point of tight image falsifying only C_i. Then the preference has to be given to the point that falsifies clauses describing the same gate. For example, in the case 3*) it is possible that there is a point $\boldsymbol{p}^*_4$ falsifying only two clauses: the clause C_3 and a clause of the original formula describing some other gate. The preference above makes one pick $\boldsymbol{p_4}$ (that falsifies clauses $v''_k \vee \sim v''_i$, $v''_m \vee \sim v''_i$ of $F(G''_i(v''_k,v''_m))$) over $\boldsymbol{p}^*_4$.

REMARK 2. When proving that $inp(T_{nat})$ contains tests detecting stuck-at faults on input line v_k of N, we employed equivalence clauses $EQ(v'_k,v''_k)$ (relating input variables v'_k and v''_k of N' and N''). If one uses the traditional definition of the miter of N' and N'' (where N' and N'' have identical input variables), the formula F is equal to $F_M \wedge z$ (no equivalence clauses are necessary). This reduces the size of R_{nat} (by the resolution operations used to "identify" variables of N' and N''). T_{nat} becomes smaller as well. As a result, one cannot guarantee that $inp(T_{nat})$ detects the testable stuck-faults on input lines of N.