A New Approach to the Use of Satisfiability in False Path Detection

Felipe S. Marques[1], Renato P. Ribas[1], Sachin Sapatnekar[2], André I. Reis[1,2]

[1]Instituto de Informática - UFRGS
CEP 91501-970 Caixa Postal: 15064
Porto Alegre – RS – Brazil
+55 (51) 3316-7036
{felipem, rpribas, andreis}@inf.ufrgs.br

[2]Department of Electrical and Computer Engineering
University of Minnesota
Minneapolis, MN 55455
(612) 625-0025
sachin@ece.umn.edu

ABSTRACT
This paper presents a novel method for false path detection using satisfiability. It is based on circuit node properties that are related to non-testable stuck-at faults as well as to false path detection. When compared to traditional satisfiability methods that generate sat instances associated to paths, the proposed method is more efficient. This efficiency derives from the fact that most digital circuits have a number of nodes that is smaller than the number of paths.

Categories and Subject Descriptors
B.6.3 [**Logic Design**]: Design Aids – *optimization, switching theory.*

General Terms
Algorithms, Design, Theory.

Keywords
Unateness, False Paths, Satisfiability.

1. INTRODUCTION
Satisfiability (SAT) is a well-known and studied computer problem [1]. Many satisfiability solvers are available, and recently the efficiency of satisfiability solvers has largely increased. For this reason, many CAD tools use a satisfiability solver as part or as the core of the problem solver. This way, many combinatorial algorithmic problems have been reduced to satisfiability instances. Satisfiability has been used for test vector generation [2] as well as for false path detection [3]. False path detection using satisfiability normally is done path by path. This means that in order to determine if a path is sensitizable, it is necessary to solve a satisfiability instance. As circuits have many paths, due to path reconvergence, the cost of using satisfiability to detect false paths is increased. This paper introduces a new method to detect false paths using satisfiability. The novelty of this method resides in the fact that the satisfiability instances to be solved are associated to properties of circuit nodes, instead of being associated to paths. This way, the proposed method is more efficient, because most circuits have a number of nodes that is smaller than the number of paths. The node property to be checked is unateness [4]. A circuit node may be positive or negative unate with respect to each of the variables on which it depends. When the node is both positive and negative unate in a given variable, it is said that the node is binate (or mixed) on that variable. It may be very easy and fast to prove that a node is binate through logic simulation. However, in order to prove that a node is not binate, it is necessary to solve a satisfiability instance. This paper compares the proposed method of using satisfiability to detect false paths based node information against the method based on path information. Main contributions of this paper are: (1) the description of a fast method to prove if a node is unate through logic simulation; (2) the method to derive sat instances to prove if a node is or is not unate; and (3) the use of this information to detect false paths.

2. SAT AND PATH SENSITIZATION
Satisfiability (SAT) input is normally expressed as a product of sums called Conjunctive Normal Form – CNF. Given a CNF representing an instance of the sat problem, a SAT solver determines a input vector for which the product is evaluated to 1, or prove that such a input vector does not exist. It is easy to produce a CNF formula for a given combinational circuit, as it is the product of the individual CNF formulas for each gate of the circuit. The CNF formula of an individual gate denotes the valid input-output assignments for the gate. CNF formulas for simple gates are shown in Table 1. They may be used to derive a SAT expression for the circuit in Figure 1.

Table 1. CNF formulas for simple gates.

gate	Equation	CNF sat
and	$o=a\cdot b$	$(a'+b'+o)\cdot(a+o')\cdot(b+o')$
nand	$o=(a\cdot b)'$	$(a'+b'+o')\cdot(a+o)\cdot(b+o)$
or	$o=a+b$	$(a+b+o')\cdot(a'+o)\cdot(b'+o)$
nor	$o=(a+b)'$	$(a+b+o)\cdot(a'+o')\cdot(b'+o')$
not	$o=a'$	$(a+o)\cdot(a'+o')$

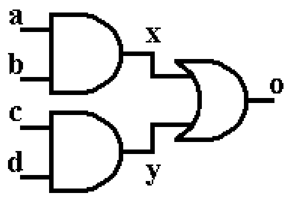

Figure 1. A simple combinational circuit.

Consider the circuit shown in Figure 1. It is composed of three simple cells, and the CNF formula for this complete circuit is obtained through the product of the CNF formulas for each gate. The circuit corresponding CNF formula is the following:

$$\begin{aligned} CNF = {} & (a'+b'+x) \cdot (a+x') \cdot (b+x') \cdot \\ & (c'+d'+y) \cdot (c+y') \cdot (d+y') \cdot \\ & (x+y+o') \cdot (x'+o) \cdot (y'+o) \end{aligned} \quad (1)$$

Satisfiability may be used to detect false paths. Consider the path from input *a* to output *o* in the circuit from Figure 1. To activate this path, it is necessary to have $y=0$ and $b=1$. Two new clauses must be added in equation 1 in order to warranty that these conditions hold. Equation 2 shows the equation 1 with the sensitization conditions added. The method used to derive the sat instance in equation 2 will be referred as **path-sat** in the remaining of this paper. Please notice that four paths are present in the circuit. Therefore, it would be necessary to create and to solve four CNF instances to test all the paths in the circuit.

$$\begin{aligned} CNF = {} & (a'+b'+x) . (a+x') . (b+x') . \\ & (c'+d'+y) . (c+y') . (d+y') . \\ & (x+y+o') . (x'+o) . (y'+o) . \\ & (y') . (b) \end{aligned} \quad (2)$$

3. UNATE AND BINATE PROPERTIES

Unateness is a property of Boolean functions and gates that is related to the polarities of the transitions they support. This property has been used to produce fast heuristic algorithms for logic synthesis, including the recursive unate paradigm used in ESPRESSO [4].

3.1 Functional unateness

Definition 1. A function $f(v1, v2, vi, ..., vn)$ is **functionally positive unate** in variable *vi* if there is a pair of input vectors $\{(a1, a2, 0, ..., an), (a1, a2, 1, ..., an)\}$ such as $f(a1, a2, 0, ..., an)=0$ and $f(a1, a2, 1, ..., an)=1$, where $ai \subset \{0, 1\}$.

Definition 2. A function $f(v1, v2, vi, ..., vn)$ is **functionally negative unate** in variable *vi* if there is a pair of input vectors $\{(a1, a2, 0, ..., an), (a1, a2, 1, ..., an)\}$ such as $f(a1, a2, 0, ..., an)=1$ and $f(a1, a2, 1, ..., an)=0$.

Definition 3. A function is said to be **functionally binate** when both definitions 1 and 2 above are satisfied for two distinct vector pairs.

Definition 4. A function is said to be **functionally independent of variable** *vi* when there is no pair of vector to satisfy neither definition 1 nor definition 2.

Notice that the above definitions for unateness are independent of the equation or the circuit implementing these functions. Next section will discuss about topological unateness, i.e. unateness related to circuit topology.

3.2 Topological unateness

Definition 5. A gate is said to be a **positive gate** if the logic function it represents is (only) positive unate with respect of all its input variables. Examples of positive gates are AND, OR and BUFFERS (non inverting buffers).

Definition 6. A gate is said to be a **negative gate** if the logic function it represents is (only) negative unate with respect of all its input variables. Examples of negative gates are NAND, NOR and inverters.

At this point it is necessary to notice that an EXOR gate is neither a positive nor a negative gate. Fortunately, this kind of gate may be decomposed in a set of simpler gates that are either positive or negative unate. Therefore, our discussion will be limited to positive and negative gates only, without loss of generality.

Definition 7. A path containing an even number of negative gates is defined as a **positive path**.

Definition 8. A path containing an odd number of negative gates is defined as a **negative path**.

Definition 9. A circuit node *n* is **topologically positive unate** in variable *vi* if there is a positive path leading from variable *vi* to node *n*.

Definition 10. A circuit node *n* is **topologically negative unate** in variable *vi* if there is a negative path leading from variable *vi* to node *n*.

Definition 11. A circuit node *n* is said to be **topologically binate** when both definitions 9 and 10 above are satisfied through two distinct paths (one positive and other negative).

Definition 12. A circuit node *n* is said to be **functionally independent of variable** *vi* when there is no path from variable *vi* to node *n*.

Functional and topological unateness may be coded as vector composed of a pair of bits $<v+, v->$ for each variable to be catalogued. The bit $v+$ assumes the value *1* when the variable is positive unate and the value *0* when the variable is not positive unate. The bit $v-$ codifies the negative unateness in a similar way. The value $<1,1>$ represents a binate node or function, while the value $<1,0>$ represents a positive unate node or function, and so forth. All the four possibilities described above may be coded this way.

Topological unateness information may be easily calculated using a recursive routine. The topological information for input variables is trivial, as all of them are topologically unate with respect to themselves and independent from others. The topological unateness at the output of a positive gate is given by the bit-to-bit OR of the unateness information at the inputs, expressed as multi-bit words. The topological unateness at the output of a negative gate is obtained in two steps. First an OR operation is made among the information from inputs, as for positive gates. Second, the positive and negative information for each variable is interchanged (*vi+*and *vi-* information is interchanged for each variable *vi*). This procedure has linear time complexity $O(|E|)$, where *E* is the edge set for the circuit graph.

Table 2 presents the bit vectors representing the topological unateness information for all the circuit nodes in Figure 2. This information may be easily obtained recursively as described previously. Notice that our procedure finds that node *out* is binate in variables *a* and *d*, positive unate in variable *b* and negative unate in variable *c*.

Table 2. Topological unateness properties of the circuit in Figure 2.

Node name	Topological unateness information							
	a+	a-	b+	b-	c+	c-	d+	d-
m	0	1	0	0	0	0	0	0
n	0	0	0	0	0	1	0	0
o	0	0	0	0	0	0	0	1
p	1	0	1	0	0	0	0	0
q	0	1	0	0	0	1	0	0
r	0	0	0	0	0	0	1	1
out	1	1	1	0	0	1	1	1

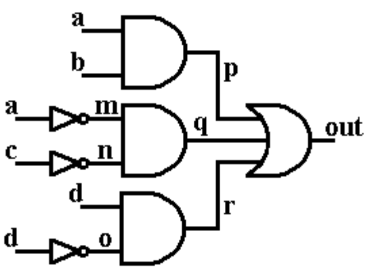

Figure 2. A combinational circuit with false paths.

3.3 Functional vs. topological unateness

It is important to notice that when expressed by the 2 bit per variable code presented in table 2, the following condition holds for the logic function represented in each node of the circuit:

functional unateness ≤ topological unateness (3)

The inequality above holds because it is not possible to a node to be functionally positive or negative binate if there is not a topological path to enable the node positive or negative unateness. However, as circuits may contain false paths, the function represented in the nodes may simply not be able to functionally use all the circuit paths. This relationship occurs for the node *out* in the circuit of Figure 2, as shown in the following.

$$\begin{array}{ccc} a{+}a{-}b{+}b{-}c{+}c{-}d{+}d{-} & \leq & a{+}a{-}b{+}b{-}c{+}c{-}d{+}d{-} \\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 0 & < & 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1 \end{array}$$

The relationship above represents the fact that the variable *d* does not affect the function in node *out*, even if there are paths from variable *d* to *out*.

4. SIMULATION AND UNATENESS

Logic simulation may be used to determine functional unateness. If the choice of the vectors is lucky, a pair of vectors where only one variable changes may prove the functional unateness of several nodes. All the nodes that have a transition will be proven to be functionally positive or negative unate, depending on the direction of the transition. However, nothing can be said about the unateness of the nodes that remained unchanged, unless an exhaustive simulation is performed, which is not practical. Anyway, the simulation of a certain number of input vectors may determine the unateness of a significant part of the circuit.

5. SATISFIABILITY AND UNATENESS

Satisfiability may be used to ensure that a circuit node is not functionally unate in a given variable. Consider the satisfiability clauses of the circuit in Figure 1. To prove that the circuit is or is not positive (or negative) unate, it is necessary to prove the impossibility of two input-output assignment pairs simultaneously (see definition 1). For this reason, the circuit clauses are duplicated. The input variable and the circuit node under test are renamed differently in the two circuits, as shown in equation 4. The same happens to internal circuit nodes, as they may have different values for the two input-output assignments. The input variables that are not being tested in the SAT instance preserve the same name, to guarantee that they will have the same value in both assignments. To test if output *o* is positive unate with respect to variable *a*, the last four clauses are necessary: *(a1'), (o1'), (a2)* and *(o2)*. The method used to derive the sat instance in equation 4 will be referred as **node-sat**.

$$\begin{aligned} CNF= {} & (a1'+b'+x1)\cdot(a1+x1')\cdot(b+x1')\cdot \\ & (c'+d'+y1)\cdot(c+y1')\cdot(d+y1')\cdot \\ & (x1+y1+o1')\cdot(x1'+o1)\cdot(y1'+o1)\cdot \\ & (a2'+b'+x2)\cdot(a2+x2')\cdot(b+x2')\cdot \\ & (c'+d'+y2)\cdot(c+y2')\cdot(d+y2')\cdot \\ & (x2+y2+o2')\cdot(x2'+o2)\cdot(y2'+o2)\cdot \\ & (a1')\cdot(o1')\cdot(a2)\cdot(o2) \end{aligned} \qquad (4)$$

6. RESULTS

Table 3 compares the number of instances to solve the false path problem based on path-sat (section 2) and our method based on node-sat (section 5). It is possible to notice that the number of instances needed to solve node-sat is significantly smaller. Another advantage of node-sat is that a significant number of sat instances may be solved through logic simulation as described in section 4. Table 4 shows the number of the node-sat instances that happen to be solved by simulation by applying 5, 25 and 125 pairs of random vectors per input variable. Notice that a number ranging from 12% to 71% of the instances were rapidly solved by simulation. Notice that this acceleration by simulation may only be performed for node-sat. In the case of path-sat, the determination of paths that are being activated through input vectors would need the maintenance and the search of the complete list of paths during simulation. This information is difficult to maintain and manage. Fortunately, for node unateness information, associated with node-sat proposed here, it is very simple to obtain and store this information, by using the two bit encoding proposed in this paper. It is only necessary to mark the nodes that switched for a pair of input vectors.

Table 3. Number of instances for path-sat and for node-sat.

Circuit	# of instances	
	path-sat	node-sat
alu2	38623	5760
C1355	4173216	24722
C1908	729056	25522
C432	291826	11828
C499	397888	21938
C3540	26603318	46644
dalu	65463	55182
i9	36980	21936

Table 4. Number of node-sat instances solved by simulation.

		Solved by simulation		
Circuit	**# of inst**	**5v**	**25v**	**125v**
alu2	5760	1751	3088	4094
C1355	24722	4750	8412	12417
C1908	25522	5136	7606	15363
C432	11828	2398	4034	5851
C499	21938	4642	8101	11633
C3540	46644	7295	19056	29026
dalu	55182	7018	13158	21817
i9	21936	4778	8336	12932

Table 5. CPU time for remaining node-sat solved by zChaff.

Circuit	# of inst	zChaff time (s)
alu2	1666	1.22
C1355	12305	1209.03
C1908	10159	384.85
C432	5977	59.41
C499	10305	3600.92
C3540	17618	9906.46
dalu	33365	728.89
i9	9004	13.81

As some sat-instances were solved through simulation, only part of the node-sat instances must be solved through a sat-solver. For example the circuit alu2 has 5760 different node-sat instances. From these, 4094 were solved by simulation. Table 5 shows the number of remaining node-sat instances and the CPU time to solve them. Notice that the number of remaining node-sat instances in table 5 is the difference given by the overall total number of node-sat instances (from table 3) minus the number of instances that were solved by simulation with 125 pairs of random vectors. The sat-solver used for the remaining instances was the zChaff solver [5]. A final issue is the percentage the difference between topological and functional unateness, as shown in table 6. The results consider unateness encoded as a pair of bits per variable. The column diff reports the number of vector positions for which functional unateness is 0 while topological unateness is 1, as described in section 3.3. These differences indicate that the circuit may contain false paths, which could be removed through a redundancy removal algorithm like [6 7].

7. CONCLUSIONS

This paper presented a new method of using satisfiability to detect false paths. This method is based on unateness information for circuit nodes and it compares functional and topological unateness of the nodes to detect the presence of false paths. This information may be used on a variety of algorithms and methods that need information about signal observability. Preliminary results show that the number of instances for unateness node detection using satisfiability is significantly smaller than for path sensitization satisfiability. These results are preliminary, and an algorithm for redundancy removal based on the method for false path detection presented here is being developed. This algorithm will implement the concept of length-disjoint paths presented in [6] as an improvement to the KMS algorithm [7].

Table 6. Difference between topological and functional unateness.

Circuit	diff	diff (%)
alu2	724	14.21
C1355	208	0.84
C1908	1250	5.11
C432	3328	32.11
C499	128	0.58
C3540	2432	5.90
dalu	11028	24.15
i9	4120	22.36
cla2	14	10.76
cla4	72	14.06
csa2_2	23	15.03
csa4_4	85	20.48
csa8_2	428	32.13
csa8_4	362	28.68
ripple2	14	12.72
ripple4	68	21.51
ripple8	296	29.13
ripple16	1232	34.52
ripple32	5024	37.83
ripple64	20288	39.67
mult2_2	12	19.35
mult4_4	209	14.11
mult6_6	521	8.99
mult8_8	968	6.65

8. ACKNOWLEDGMENTS

This research was partially supported by CNPq, CAPES, and FAPERGS Brazilian Founding Agencies.

9. REFERENCES

[1] Marques-Silva, P.J., and Sakallah, K.A., Boolean Satisfiability in Electronic Design Automation, *Design Automation Conference*, 2000, pp. 675-680.

[2] Larrabee, T., Test pattern generation using Boolean satisfiability, *IEEE Transactions on CAD*, Vol. 11, Issue: 1, January 1992, pp. 4 – 15.

[3] Ringe , M., Lindenkreuz, T., and Barke, E., Path verification using Boolean satisfiability, *Design, Automation and Test in Europe*, Feb. 1998, pp. 965 – 966.

[4] Brayton, R.K., Hachtel, G.D., McMullen, C.T., and Sangiovanni-Vincentelli, A.L.M., *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, Boston, 1984.

[5] Fu, Z., Mahajan, Y., and Malik, S., zChaff Solver, http://www.princeton.edu/~chaff/zchaff.html

[6] Saldanha, A., Brayton, R.K., and Sangiovanni-Vincentelli, A.L.M., Circuit structure relations to redundancy and delay, *IEEE Transactions on CAD*, Vol. 13, issue 7, July 1994, pp. 875 – 883.

[7] Keutzer, K., Malik, S., and Saldanha, A., Is redundancy necessary to reduce delay?, *IEEE Transactions on CAD*, vol. 10, issue 4, 1991, pp. 453-469.