Differentiating Diagnostic Theories through Constraints over an Eight-valued Logic

Francisco Azevedo & Pedro Barahona[1]

Abstract. In this paper we address the issue of diagnosing propositional theories where only some components are observable. More specifically, the goal is to find tests that allow the differentiation of two alternative theories. This subject is addressed in the context of differential diagnosis of faulty gates in a VLSI circuit where the only observable findings are its input/output bits, but may be extended to other areas, namely in diagnosing a theory of an agent where knowledge about it is based on its response to external stimuli.

To model these problems we developed an eight-valued logic that describes the dependency of the findings on the competing theories. Additionally, we implemented a constraint solver to handle this eight-valued logic domain, and to solve efficiently the problem of obtaining differentiating tests that allow the elimination of one of the alternative theories. We discuss the limitations of the currently more advanced techniques to handle disjunctive constraints, and propose a new method, iterative time-bounded search (ITBS) to overcome them. The method is exemplified and tested in the problem of generating differential test patterns for digital circuits.

1. INTRODUCTION

When the output of some system does not correspond to its expected behaviour for a given input, one is faced with the problem of diagnosis. In this paper we are particularly concerned with systems modelled by means of a set of propositional rules where only the systems' input and output may be observed. This means that although we may assume alternative components for the system model, we may not, in general, directly monitor them because they are not accessible. This is the case dealt with in this paper, VLSI combinational circuits, where the propositional theory is embodied in the circuit components (gates) but only the input and output bits are accessible for monitoring. Nonetheless, the approach is more general and can be adapted to other situations (e.g. an agent whose models can only be tested by observing its behaviour for some inputs).

In this context, diagnosing a system requires the generation of some specific input that makes the faulty component visible at the output. We denote such input as an input test pattern, following the usual VLSI terminology, where this is known as the Automatic Test Pattern Generation (ATPG) problem. Current techniques deal with this problem by trying to generate input vectors that cause different outputs in two circuits (among the normal and the alternative abnormal circuits) [6,5,10]. Although there are techniques to avoid duplicating the whole circuit, the complexity of the diagnosis increases significantly with the extra circuit.

Alternatively, a technique for generating test patterns for VLSI circuits was suggested in [12] to code the dependency of a digital signal on the (faulty) state of a gate: the Boolean domain that models digital circuits was extended with two extra values that denote such dependencies. A test pattern is thus an input yielding one such extra value at some output bit.

We have extended this idea further to differentiate two alternative models, and introduced an 8-valued logic whose values not only denote dependency on faulty gates, but also discriminate the dependencies between two sets of faulty gates [3]. In diagnosis, there might be two sets of faults that, given some input, yield the same (faulty) output. Hence, we are interested in generating differential input test patterns, which may differentiate between two alternative models. A differential test pattern is then an input of the circuit that produces an output containing one of the values that discriminate between the two diagnostic sets of arbitrary size. Despite being inspired in that 8-valued logic, a previous system we developed was implemented with a "traditional" 0/1 Boolean solver.

In this paper we present a specialised constraint solver which, by handling this 8-valued logic directly, greatly improves the average performance of our previous system to solve problems in this domain. Moreover, the problem of differentiating alternative models imposes that some specific values appear in any one of the outputs of the system (e.g. the VLSI circuit). This corresponds to a disjunctive constraint, which in many cases cannot be handled appropriately even by the currently most advanced techniques available in constraint programming, namely the cardinality constraint [13]. In the paper we thus propose a new technique,

[1] {fa,pb}@di.fct.unl.pt
Departamento de Informática, Universidade Nova de Lisboa
2825-114 Caparica — Portugal

iterative time-bounded search (ITBS), to overcome these limitations.

The paper is organised as follows. Section 2 presents the 8-valued logic. Section 3 shows how it can be used to model the generation of differential test patterns in combinational circuits. Section 4 describes a constraint solver to handle directly the 8-valued logic. Section 5, introduces and justifies the ITBS method. Experimental results obtained in the generation of differential test patterns are discussed in section 6, before the conclusions are summarised in the last section.

2. THE 8-VALUED LOGIC

The purpose of defining our 8-valued logic is to be able to trace the dependency of a truth value on the alternative theories that are to be differentiated. It is important to notice the context in which this differentiation takes place.

Initially, there is a model *N* of a circuit (or some more general system) that is assumed to be correct. When an output *O* is observed which is not consistent with the input *I* and the circuit model, such output can be explained by alternative theories that justify the anomalous finding. Let us consider two such theories, T_1 and T_2 whose changes regarding theory *N* consist of some (faulty) components, denoted by F_1 and F_2 respectively ($F_1 \subseteq T_1$, $F_2 \subseteq T_2$) and that given input *I* logically entail output *O*. Since both theories logically entail, from input *I*, an output *O* which is different from that entailed by the initial model *N*, *I* is an input test pattern for both F_1 and F_2. However, *I* does not enable the differentiation between F_1 and F_2, since both produce the same output for input *I*. We are thus concerned with finding a differential test pattern *D*, that given as input to theories T_1 and T_2, yields two different outputs, O_1 and O_2.

In case of a digital circuit, outputs O_1 and O_2 will be different if at least one of the output bits is different, and this means that such bit depends on either F_1 or F_2, but not on both. The logic presented in this section captures this notion of dependency by means of its 8 values. Each value is denoted by a pair <*p*-*X*>: *p* ranges over the set {*n*, *m*, d_1 and d_2} and denotes the dependency on the alternative theories; *X* ranges over the usual 0/1 truth values and is the "physical" truth value that could eventually be observed in the behaviour of the agent with the initial model *N*. The intuitive meaning of the 8 values is thus the following:

n-X	the truth value is independent from faults F_1 and F_2. The faults to the initial theory *N* have no influence on the physical truth value which is always *X*.
d_1*-X*	the truth value depends on faults F_1 but not on F_2. If F_1 occurs, the physical truth value is the complement of *X*; otherwise it is *X*, as yielded with *N* and theory with F_2.
d_2*-X*	similar to the previous, with F_1 and F_2 swapping roles.
m-X	the truth value depends on both faults F_1 and F_2. If any of them occurs, the physical truth value is the complement of *X*.

With the above definitions let us analyse the problem of encoding, in general, the truth value *Z* of some proposition in this 8-valued logic. Table 1 shows the 8 possible cases.

Table 1. Truth value of a proposition with a normal model and two different theories

N	0	0	0	0	1	1	1	1
T_1	0	0	1	1	0	0	1	1
T_2	0	1	0	1	0	1	0	1
Z	n-0	d_2-0	d_1-0	m-0	m-1	d_1-1	d_2-1	n-1

In the first column of the table, the physical truth value is the same (0) in all cases, i.e. when either one or none of the theories T_1 and T_2 are considered. In the second column, the physical truth value is 0 in the initial theory *N* and in theory T_1, but 1 in theory T_2. It thus depends on faults F_2 (but not on F_1) and is encoded as d_2*-0*. In the fourth column, the physical truth value is 0 in the initial theory *N* but 1 in T_1 and T_2. It is thus dependent on both F_1 and F_2 and encoded as *m-0*.

2.1 Boolean operations

A similar analysis can be performed to define the semantics of the usual Boolean operations. For example, the semantics of the 8-valued logic exclusive disjunction (xor) is shown in Table 2.

Table 2. XOR truth table in 8-valued logic

xor	**m-0**	**d_2-0**	**d_1-0**	**n-0**	**n-1**	**d_1-1**	**d_2-1**	**m-1**
m-0	n-0	d_1-0	d_2-0	m-0	m-1	d_2-1	d_1-1	n-1
d_2-0	d_1-0	n-0	m-0	d_2-0	d_2-1	m-1	n-1	d_1-1
d_1-0	d_2-0	m-0	n-0	d_1-0	d_1-1	n-1	m-1	d_2-1
n-0	m-0	d_2-0	d_1-0	n-0	n-1	d_1-1	d_2-1	m-1
n-1	m-1	d_2-1	d_1-1	n-1	n-0	d_1-0	d_2-0	m-0
d_1-1	d_2-1	m-1	n-1	d_1-1	d_1-0	n-0	m-0	d_2-0
d_2-1	d_1-1	n-1	m-1	d_2-1	d_2-0	m-0	n-0	d_1-0
m-1	n-1	d_1-1	d_2-1	m-1	m-0	d_2-0	d_1-0	n-0

As expected, xor-ing independent values (*n*-0/1) produces independent values according to the usual xor truth tables (centre of Table 2). Xor-ing a d_1 signal (truth value) and a d_2 signal results in an *m* signal reflecting the dependency on both F_1 and F_2. Due to the nature of the exclusive disjunction, when both inputs depend on the same faults, the output of the gate does not depend on any, and is thus an *n* signal. Perhaps more interestingly, xor-ing an *m* signal (dependent on both sets of faults) with a signal that only depends on a set of faults, makes the output solely dependent on the other set of faults.

This latter case is explained in Table 3 below. Columns *N*, T_1 and T_2 represent the cases to consider. The first two lines represent the physical truth values which are to be xor-ed: input signal *m-1*, takes the physical truth value 1 in theory *N*, but 0 in theories T_1 and T_2, since it depends on both faulty components, as an *m* signal; input signal d_1*-0*, takes the physical truth value 0 in theories *N* and T_2, but 1 in theory T_1 (it depends only on F_1 as a d_1 signal). The physical truth value of the output is obtained by xor-ing the physical truth values of the inputs. Since the physical output is 1 in both theories *N* and T_1, but 0 in T_2 it is coded as d_2*-1*.

Table 3. Physical output of an xor-gate with inputs *m-1* and d_1*-0*

	N	T_1	T_2
m-1	1	0	0
d_1-0	0	1	0
m-1 $\oplus$ d_1-0	1	1	0

Similar reasoning was used to define the semantics of all the usual Boolean operations implemented by digital gates for the extended 8-valued logic.

3. MODELLING ALTERNATIVE DIAGNOSTIC THEORIES IN DIGITAL CIRCUITS

We now show how to model a faulty digital circuit for which there are two alternative diagnoses available. As usual we will assume that the faults correspond to some of the circuit gates being either stuck-at-0 or stuck-at-1. This is equivalent to inserting an additional buffer at the output of the faulty gates and making this buffer faulty. By doing so, it is only necessary to model one kind of faulty gate, the buffer, leaving the rest of the circuit unchanged. These buffers are referred to below as S-buffers.

According to the notation of the previous sections, theory *N* corresponds to the circuit where all the S-buffers are functioning correctly. In diagnostic theory T_i, (i in {1,2}), some gates are stuck (corresponding to the F_i component of theory T_i) and the output of the corresponding S-buffers have a fixed physical truth value. Table 4 below shows the resulting model for all S-buffers.

Table 4. Truth table for the 8 different types of S-buffer

Input= **Dependency**	**m-0**	**d_2-0**	**d_1-0**	**0**	**1**	**d_1-1**	**d_2-1**	**m-1**
T_1/0, T_2/0	n-0	n-0	n-0	n-0	m-1	m-1	m-1	m-1
T_2/0	d_1-0	n-0	d_1-0	n-0	d_2-1	m-1	d_2-1	m-1
T_1/0	d_2-0	d_2-0	n-0	n-0	d_1-1	d_1-1	m-1	m-1
T_1/0, T_2/1	**d_2-0**	d_2-0	d_2-0	d_2-0	d_1-1	d_1-1	d_1-1	d_1-1
T_1/1, T_2/0	d_1-0	d_1-0	d_1-0	d_1-0	d_2-1	d_2-1	d_2-1	d_2-1
T_1/1	m-0	m-0	d_1-0	d_1-0	n-1	n-1	d_2-1	d_2-1
T_2/1	m-0	d_2-0	m-0	d_2-0	n-1	d_1-1	n-1	d_1-1
T_1/1, T_2/1	m-0	m-0	m-0	m-0	n-1	n-1	n-1	n-1

The table entries are explained with an example. The fourth line corresponds to an S-buffer that is stuck-at-0 in diagnostic theory T_1 and stuck-at-1 in theory T_2, and the first entry in that line corresponds to its output, given an input signal *m-0* (i.e. being an *m* signal, the input of the gate denotes that it depends on both faults F_1 and F_2, i.e. on other faulty gates in theories T_1 and T_2). This case is depicted in Table 5.

Table 5. Output of an S-buffer (stuck-at-0 in T_1 and stuck-at-1 in T_2) for input *m-0*

	N	T_1	T_2
m-0	0	1	1
S-buffer (T_1/0,T_2/1) output	0	0	1

The first line corresponds to the physical truth value that is input to the buffer, which is 0 in theory *N* and 1 in both theories T_1 and T_2. In case of theory *N* (no faults) the buffer behaves correctly and outputs the same value of the input. In theory T_1 the buffer is stuck-at-0 and thus outputs 0. Finally, in theory T_2 the buffer is stuck-at-1 and thus outputs 1. Inspection of the output, shows that it is 0 in both theories *N* and T_1, but 1 in theory T_2; it is thus coded as *d_2-0*.

4. A CONSTRAINT SOLVER FOR AN 8-VALUED LOGIC

In a previous approach [3], despite using an 8-valued logic, we addressed the problem of differential diagnosis of digital circuits with a 0/1 Boolean solver (the extra 6 values were propagated to the output, by analogy with the demons approach taken in [6] but with choice points to cope with the different possibilities) implemented in SICStus [11]. The basic constraints 'not', 'and' (with an arbitrary number of inputs) and 'xor' were modelled as user-defined constraints and the others (e.g. 'or' and 'nand') modelled in terms of the basic ones.

To implement the solver, not only was necessary to adapt the truth tables of the usual Boolean operators (plus the faulty gates) for the particular encoding of the 8-valued logic, but also to specify a constraint propagation strategy.

All user-defined constraints that implement this 8-valued logic were thus activated by guards, normally fired at instantiation of one of its variables. In addition to these higher-level constraints, cardinality constraints were used, as well as constructive disjunction and conjunction for some relations between 2 variables. By so doing, the constraint solver imposes a form of consistency stronger than node consistency, but somewhat weaker than arc consistency (too costly to maintain). Though possibly not optimally, we think that the most interesting cases for constraint propagation were considered.

For example, when one of the variables in a binary 'xor' constraint takes value *n-0* (the guard checks this condition) the constraint is simply rewritten as an equality of the other two variables. As another example, if one variable takes value *m-1*, there are eight possible pairs of values for the other two variables, and the 'xor' constraint is thus rewritten into the constructive disjunction of these cases. In a final example, if the output of an 'and-gate' takes value *n-0*, the 'and' constraint is fired and may be replaced by the cardinality constraint imposing that at least one of the inputs be *n-0* (in fact, there are some combinations of d_i and *m* signals that also yield value *n-0* when 'anded'; the 'and' constraint checks this possibility).

5. ITERATIVE TIME-BOUNDED SEARCH

As explained in the introduction, to be a differential test pattern, an input must produce a d_i signal in one of the output bits of a digital circuit. This is a typical disjunctive constraint (either in the first output bit, or in the second, or the third, …). A classical technique to efficiently handle such constraints consists of delaying the choice of the alternatives until a final labelling eventually makes a commitment to one of the disjuncts. Such technique, using the cardinality operator [13], is usually much more efficient than traditional depth-first search, since making an early mistake can be very costly to undo.

However, the least commitment strategy has a price: the problem is kept less constrained than that resulting from making an early commitment. In particular, less propagation is usually possible, or only weaker heuristics may be used in the less constrained problem.

The problem with early commitment is thus the difficulty in undoing the wrong choices. But often, there is a huge difference in complexity of solving the different problems that result from making each of the possible choices. Some of these, might also be much more easy to solve than the whole problem. If this is the case, it pays to take some time trying to solve each of the sub-problems in a round-robin discipline. To make the strategy complete, the time allowed for each of the tries may be increased in subsequent round-robin turns. This is the basic idea behind the iterative time-bounded search (ITBS) that we propose for this kind of problems and applied for the specific case of finding differential test patterns.

5.1 The method

The method may be more completely described as follows. Being the initial problem composed of a disjunction with k disjuncts, ITBS assigns a time limit T to solve each of the sub-problems defined with a commitment to one of the disjuncts. If all sub-problems were aborted due to exceeding time limit T, this limit is doubled (more generally multiplied by some factor f). The time is thus increased in each of the rounds, until a solution is found.

ITBS thus shares the underlying idea of iterative deepening and other techniques (e.g. [7], [9], [14]) to overcome the problems of depth-first search, by searching side-branches before fully exploring a previously selected branch of the search space.

5.2 Complexity Analysis

If a solution is found in round r ($r \geq 0$), the worst execution time of ITBS is $A = k*(T+T*f+\ldots+T*f^{\,r})$, occurring if the best choice is the last. Compared with the best execution time achieved when committing to the right choice but with no time limit, i.e. $B=T*f^{r-1}$, ITBS is penalised by a factor of

$$A / B = [k*T*(f^{\,r+1}-1)/(f-1)] / (T*f^{\,r-1}) \approx k*f$$

If B is much less than time C obtained when solving the full problem with a least commitment strategy (more precisely, when $B < C / (k*f)$), the ITBS strategy pays off. ITBS thus pays off whenever there is some heuristic to solve one sub-problem much faster than the whole problem (with least commitment). This is, of course, problem dependent. We tested this strategy in the problem of finding differential test patterns.

5.3 Heuristics

To do so, we tried three different scenarios. In the first, we used the least commitment approach, implemented with a cardinality operator on the output bits that included a d value in their domains after setting up the circuit. To speed up this execution, we labelled first the input bits that lead to the possible S-buffers, and then the inputs that lead to the possible output bits. This is the Cardinality (#) heuristic.

The second scenario adopted the ITBS strategy with a heuristic that is similar to the previous one, but which is applied only after committing to one of the output bits. This is the ITBS-Bit strategy. The advantage now is that by committing to some output bit, more propagation is possible in principle and less input bits are relevant, which thus decreases the search space.

The third scenario, ITBS-Path, also used the ITBS technique but with a different heuristic (Path) that is obtained after selecting a definite d-signal to an output bit. Since this d-signal depends either on faulty components F_1 (values d_1-0 and d_1-1) or on faulty components F_2 (d_2-0 and d_2-1), it is then mandatory that the corresponding S-buffers produce the d signal. This is imposed with a cardinality constraint. To reach the output bit, there must be a differential test path, i.e. a path from an S-buffer to that bit [3]. We then label the variables starting in the d-signal "backwards" to an S-buffer in a way that they remained always dependent on F_1 or F_2. We then label the input bits that are relevant to the path, i.e. that allow the d signal to reach the output bit. We did it starting at the S-buffer (the path start gate) and proceeding backwards through its dependencies, and eventually reaching some input bits, which are then labelled first. We then followed along the chosen differential test path. For each gate in the path, we would similarly find the input bits of the circuit that are relevant for this gate, and label them in the second place. This process finishes when reaching the final path bit (a circuit output).

6. EXPERIMENTAL RESULTS

To test the performance of our system, we used the ISCAS set of benchmark digital circuits [8], widely used by the Electronics Computer Aided Design community for testing several digital circuit design tools and techniques.

Table 6. ISCAS benchmark circuits

circuit	# gates	#in bits	#out bits	circuit	# gates	#in bits	#out bits
432	196	36	7	**2670**	1426	233	140
499	243	41	60	**3540**	1719	50	22
880	443	60	26	**5315**	2485	178	123
1355	587	41	32	**6288**	2448	32	32
1908	913	33	25	**7552**	3719	207	108

For the problem of differentiating fault diagnoses we created, for each circuit, some specialised benchmarks [2]. Each benchmark consists of a set of all minimal diagnoses that explain some faulty behaviour of a circuit. These are the most interesting diagnoses to differentiate as they are tightly related. For each problem, the number of variables is the number of gates plus the number of input bits, and the number of constraints is roughly the double of the variables.

Table 7 below shows some of the experimental results obtained with our solver, a system implemented in SICStus over Linux on a PentiumIII/500 (times reported are all in seconds).

Table 7. Differentiation results with the different heuristics

						ITBS							
				#		Bit				Path			
circuit	F1	F2	No	Time	Ni	bit	Lim	TT	Ni	d	Lim	TT	Ni
432	47/1,430/0	270/1,430/0	3	n/a	36	3	5	10.2	36	2	5	5.2	36
432	223/0,338/1	223/0,319/0	4	396.4	36	2	5	5.3	36	2	5	5.2	36
432	223/0,430/1	223/0,338/1	4	0.2	36	2	5	5.3	36	2	5	5.3	36
432	223/0,386/1	223/0,319/0	4	373.0	36	2	5	5.2	36	2	5	5.2	36
432	37/1,105/0	270/1,430/0	7	n/a	36	1	5	0.3	19	1	5	0.3	18
432	329/0,430/0	270/1,430/0	5	0.2	36	1	5	0.2	27	1	5	0.2	27
6288	3486gat/0	2434gat/1	23	4.2	32	1	5	4.1	20	3	5	14.0	20
6288	5348gat/1	5163gat/1	7	4.8	32	1	5	4.8	32	2	10	74.8	32
6288	5461gat/0	4808gat/1	7	4.8	32	1	5	4.8	32	5	5	24.9	32
6288	6285gat/0	5727gat/1	2	5.3	32	1	5	5.8	32	1	5	5.7	32
6288	1173gat/0	1128gat/0	17	4.8	32	1	5	4.6	32	1	5	4.8	32
6288	1546gat/1	1343gat/1	23	n/a	32	n/a	n/a	n/a	n/a	1	5	4.0	20

The two sets of faults F_1 and F_2 that we want to differentiate consist of one or more faults in the form *gate/stuck-at-value*; N_o is the number of output bits that remain with d-values in their domains after all the gate constraints have been posted and propagated; N_i is the number of input bits that must be labelled to guarantee the solution; *TT* is the total time needed to obtain the solution with ITBS, being *Lim* the time limit when it was found; *bit* indicates on which of the possible output bits the solution was found; and *d* indicates the number of the successful d-value choice. Non available values represent aborted executions.

6.1 Discussion

The cardinality operator to deal with disjunctive constraints is often effective. However, in a significant number of cases the results are quite disappointing. In the smaller circuit, *c432*, the first test shown could not be solved in any acceptable time limit (tens of hours). In fact, by not committing to any of the 3 output bits where a d signal occurs, no significant pruning of the search space (2^{36}) was achieved. This is not the case with the ITBS method where such commitment is enough to propagate enough information to prune the search space. Of course, the most effective choice was not always considered first. But here the time bounded nature of the ITBS method avoids a strong commitment to the wrong choice. For example, ITBS with the Bit heuristic easily finds a solution in the 3rd choice, being interrupted after 5 seconds of fruitless search in the previous 2 choices. Similarly, ITBS with the Path heuristic finds a solution with the second choice of a *d* signal.

It is not simple to analyse how the pruning takes place. In general, it is due to constraint propagation, as the number of bits to label is the same in all heuristics. There are some exceptions though. In one of the experiences with the *c432* circuit, committing to some output bit immediately reduces the number of bits to label from 36 to 19 (ITBS-Bit) or 18 (ITBS-Path) with the corresponding efficiency improvement.

Of course, a heuristic is not always guaranteed to succeed. Even when there is a commitment to some output bit (and hence a stronger heuristic (Bit) compared to the case with no commitment (#)), this stronger heuristic is not guaranteed to find a solution (e.g. last line in Table 7). However, using the other more specialised heuristic (Path) one was always able to find a solution (sometimes only at the cost of being diverted from the wrong choices by the time limits of ITBS).

This shows the importance of the ITBS technique as an alternative to the cardinality operator. The former allows the use of more powerful heuristics in the labelling phase of problem solving. The risks associated to the wrong choices are softened by the time bounded commitment to them.

7. CONCLUSIONS AND FURTHER RESEARCH

In this paper we presented an 8-valued logic and showed how it could be used to differentiate between alternative diagnostic theories in the cases where the initial theory is not fully observable. The methodology is illustrated in the problem of generation of differential test patterns in combinational circuits.

This problem presents a rather large search space, and to solve it efficiently we developed a specialised constraint solver for this 8-valued logic. This solver appears to be quite effective to handle these problems but its efficiency has still to be properly assessed (it should be compared with one that could be automatically generated from the specification of our logic [1]).

The disjunctive nature of some constraints in the problem pushed to the limit the usual techniques to handle these constraint solving problems, namely the cardinality operator. We therefore proposed a new technique, iterative time-bounded search, that aims at avoiding the dramatic consequences of bad initial choices in the depth-first search used in the labelling phase of constraint solving.

So far we have not compared our results with those obtained in the ECAD area ([6,5,10]), but we intend to do it soon. We expect that our modelling technique, by using only one circuit rather than duplications of the circuit, may be competitive with the current approaches in this area, even if requiring some tuning of the existing constraint solver (this hypothesis will be tested in a joint project with colleagues from this area). Another interesting result, lies in the possibility of extending our modelling to other ATPG problems, such as optimisation problems (e.g. generation of test patterns that detect a maximum number of faults). Rather that using more values to represent dependencies, the signals could carry set variables, denoting the sets of faults they depend on. This modelling would therefore rely on the use of set constraints [4].

ACKNOWLEDGEMENT

The first author was financially supported by "Sub-Programa Ciência e Tecnologia do 2º Quadro Comunitário de Apoio".

REFERENCES

[1] Krzysztof R. Apt and Eric Monfroy. *Automatic Generation of Constraint Propagation Algorithms for Small Finite Domains*, in Proceedings of CP'99, Joxan Jaffar (Ed.), Springer, pp. 58-72, 1999.

[2] F. Azevedo and P. Barahona. *Benchmarks for Differential Diagnosis*, at URL htttp://www-ssdi.di.fct.unl.pt/~fa/differential-diagnosis/benchmarks.html.

[3] F. Azevedo and P. Barahona. *Generation of Test Patterns for Differential Diagnosis of Digital Circuits (Extended Abstract)*, in Proceedings of CP'98, M. Maher and J.-F. Puget (Eds.), Springer, p. 462, 1998.
Long version in Proceedings of the 1998 ERCIM/COMPULOG Workshop on Constraints, K. Apt, P. Codognet and E. Monfroy (Eds.).

[4] C. Gervet, *Interval Propagation to Reason about Sets: Definition and Implementation of a Practical Language*, Constraints International Journal, vol. 1, Number 3, Kluwer Academic Publishers, pp.191-244, March 1997.

[5] T. Gruning, U. Mahlstedt, H. Koopmeiners, *DIATEST: A Fast Diagnostic Test Pattern Generator for Combinational Circuits*, Proceedings of the IEEE International Conference on Computer-Aided Design (ICCAD91), pp. 194-197, 1991

[6] I. Hartanto*1, V. Boppana, W.K. Fuchs, J.H. Patel, *Diagnostic Test Pattern Generation For Sequential Circuits*, Proc. 15th VLSI Test Symposium (VTS), Monterey, pp. 196-202, April 1997.

[7] W. D. Harvey and M. L. Ginsberg. *Limited discrepancy search*, in Proc. of the 14th Int. Joint Conf. on A.I., 1995.

[8] ISCAS. *Special Session on ATPG*, Proceedings of the IEEE Symposium on Circuits and Systems, July 1985.

[9] P. Meseguer. *Interleaved Depth-first search*, in Proceedings of the 15th Int. Joint Conf. on A. I., pp. 1382-1387, 1997.

[10] I. Pomeranz, S.M. Reddy, *A Diagnostic Test Generation Procedure for Synchronous Sequential Circuits based on Test Elimination*, International Test Conference (ITC98). Washington, D.C., USA, pp. 1074-1083, 1998.

[11] Programming Systems Group of the Swedish Institute of Computer Science. *SICStus Prolog User's Manual*, 1995.

[12] H. Simonis. *Test Generation using the Constraint Logic Programming Language CHIP*, in Proc. of the Sixth Int. Conf. on Logic Programming, MIT Press, pp 101-112, 1989.

[13] P. Van Hentenryck and Y. Deville. *The Cardinality Operator: a new logical connective and its application to constraint logic programming*, in: Eighth International Conference on Logic Programming, 1991.

[14] T. Walsh. *Depth-bounded discrepancy search*, in Proc. of the 15th Int. Joint Conference on Artificial Intelligence, 1997.