Efficient Model-Based Diagnosis of Sequential Circuits

Alexander Feldman,[1] Ingo Pill,[2] Franz Wotawa,[2] Ion Matei,[1] Johan de Kleer[1]
[1]Palo Alto Research Center Inc., 3333 Coyote Hill Road, Palo Alto, CA 94304, USA
[2]Institute for Software Technology, Graz Univ. of Technology, Inffeldgasse 16b/II, 8010 Graz, Austria
{afeldman, imatei, dekleer}@parc.com, {ipill, wotawa}@ist.tugraz.at

Abstract

In Model-Based Diagnosis (MBD), we concern ourselves with the health and safety of physical *and* software systems. Although we often use different knowledge representations and algorithms, some tools like satisfiability (SAT) solvers and temporal logics, are used in both domains. In this paper we introduce Finite Trace Next Logic (FTNL) models of sequential circuits and propose an enhanced algorithm for computing minimal-cardinality diagnoses.

Existing state-of-the-art satisfiability algorithms for minimal diagnosis use Sorting Networks (SNs) for constraining the cardinality of the diagnostic candidates. In our approach we exploit Multi-Operand Adders (MOAs). Based on extensive tests with ISCAS-89 circuits, we found that MOAs enable Conjunctive Normal Form (CNF) encodings that are significantly more compact. These encodings lead to 19.7 to 67.6 times fewer variables and 18.4 to 62 times fewer clauses. For converting an FTNL model to CNF, we could achieve a speed-up ranging from 6.2 to 22.2. Using SNs fosters 3.4 to 5.5 times faster on-line satisfiability checking though. This makes MOAs preferable for applications where RAM and off-line time are more limited than on-line CPU time.

Introduction

Model-Based Diagnosis (MBD) is a subdiscipline of Artificial Intelligence (AI) where we study how systems fail (Reiter 1987; de Kleer and Williams 1987). Model-checking (Clarke, Grumberg, and Peled 1999), with its origins in logic and formal verification, is a topic with application to AI where we develop languages and methods for proving system properties. In MBD, we use a model of a system's correct specification and derive diagnoses as fault sets that offer explanations for what happened in the system for given observations of abnormal behavior. In model-checking, we instead use a model of a possibly incorrect system and ask whether it complies to the specification—deriving a proof or a counterexample.

What makes model-checking and MBD close is that instead of a large data set, we reason with a formally specified model. In MBD, we often abstract a system's dynamic behavior away, in order to (1) make the modeling task simpler, and (2) to prevent a combinational blow-up. Model checking, on the other hand, is preoccupied with the temporal aspects of a model. As a result, we can exploit model-checking methods, languages, and tools for the advancement of MBD.

In this context it was shown, e.g., how to exploit Linear-Temporal Logic (LTL) (Pnueli 1977) for diagnostic purposes and diagnose corresponding models (Pill and Quaritsch 2013). To the best of our knowledge, all existing MBD approaches based on temporal logics make assumptions about the temporal nature of faults. Examples are constant intermittency (Abreu, Zoeteweij, and van Gemund 2009) or persistence (Pill and Quaritsch 2013). These assumptions greatly simplify the computational complexity of reasoning but are not realistic, for example, in diagnosing soft-failures in high-radiation environments such as space. In physical environments periods of intermittent failures are exceptionally hard to predict. Our diagnostic framework is based on Finite Trace Next Logic (FTNL). It is completely general: any persistent, periodic intermittent, or aperiodic intermittent behavior is going to be diagnosed.

We furthermore propose a novel SAT-based diagnosis algorithm. Since MBD is closer to partial MaxSAT (Tompkins and Hoos 2004) than to SAT, we have to encode cardinality constraints (Asín et al. 2011; Nica et al. 2013). For this, we analyze the performance trade-offs of binary constraints based on digital multipliers and sorting networks.

Our concepts and algorithms help in understanding why some behavior violates a temporal model, like failed test cases or counter-examples from model-checking. Furthermore we demonstrate a direct practical application of our algorithms in analyzing faults in Integrated Circuits (ICs) such as Field-Programmable Gate Arrays (FPGAs) as, e.g., resulting from radiation in space (Petersen 2011). An improved understanding of how individual gates might fail in space and in time will allow us to create robust, and possibly even fail-safe designs that are immune to transient failures.

Our research contributes to the field of MBD in multiple ways. (1) We propose a new framework using FTNL as a simple temporal logic for diagnosing synchronous sequential logic where we do not make assumptions about the temporal behavior of multi-faults; (2) We propose a SAT-based algorithm with a novel, more compact encoding for FTNL diagnosis that outperforms state-of-the-art MBD SAT-based algorithms in our tests; (3) We evaluate our approach for IS-

CAS-89 synchronous circuits (Brglez, Bryan, and Kozminski 1989) with multiple-faults and characterize the scalability of our algorithms.

Finite Trace Next Logic and Diagnostics

Synchronous sequential circuits extend combinational (Boolean) circuits by introducing memory elements such as flip-flops. In practice, D-type flip-flops are very attractive, since we can implement them with a small number of transistors on one hand, and on the other hand we can use them to implement also other memory element types.

Figure 1 shows an n-bit counter with D-type flip-flops in standard VLSI notation (Parhami 2009). The ability to generate counters of various size by setting n is important for analysis, benchmarking and for empirically studying algorithmic trade-offs. As a running example we use a 2-bit counter ($n = 2$). The 2-bit counter contains two D-type flip-flops (f_1, f_2), one AND-gate (a_2), and two XOR-gates (x_1, x_2).

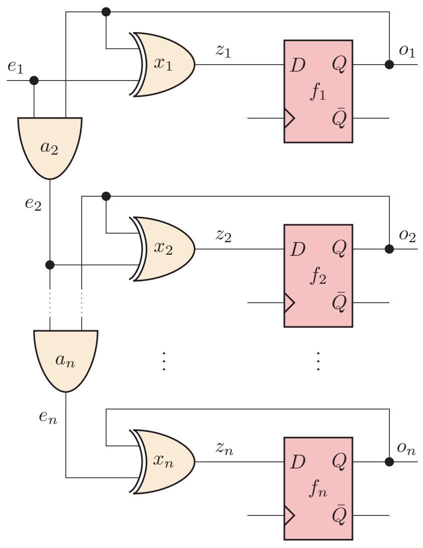


Figure 1: n-bit counter

We assume the system to have a single synchronous clock. Since we can redesign any multi-clock synchronous system to use a single clock (using counters for deriving the individual clocks) this is not a real issue. In principle we could extend the approach to support also asynchronous clocks, which is out of the scope of this paper though.

Finite Trace Next Logic

There are several logics like LTL (Pnueli 1977), PSL (Eisner and Fisman 2006), or FLTL (Pill and Wotawa 2018) that have been successfully used to define specifications of sequential circuits and many other systems. With our focus on the circuits themselves (or describing implementations in general), we do not need temporal operators like the until (Pnueli 1977). Thus, and in order to support a compact presentation, we use a simpler logic that considers *finite* traces as experienced in practice, and which has only one temporal operator—for referring to the *next* step. Compared to, e.g., FLTL and as can bee seen in Def. 3, we use the weak semantics for this operator (see (Pill and Wotawa 2018) for a discussion of strong and weak FLTL semantics).

While LTL is best suitable for proving properties of systems, handling infinite loops, and supporting various types of queries such as dead-locks and critical states, FTNL targets reasoning about digital systems with state. Of course, it is possible to use LTL for the same task, but the support of unused operators like *until* comes at a computational price.

Definition 1 (FTNL Syntax)**.** Let V be a finite set of propositional variables. An FTNL formula is then defined inductively as follows:

1. for any $p \in V$, p is an FTNL formula;
2. if φ and ψ are FTNL formulas, then $\neg\varphi$, $\varphi \vee \psi$, and $\mathbf{X}\,\varphi$ are also FTNL formulas.

The *alphabet* $\Sigma = 2^V$ describes variable assignments, and finite sequences of *letters* $\sigma \in \Sigma$ can be used to describe the finite behavior of a system. We refer to these sequences also as *finite words* or *traces*.

Definition 2 (Finite Trace)**.** Given a set of propositional variables V, a *finite trace* $\tau = \tau_1, \tau_2, \ldots, \tau_n$ is a sequence of letters $\tau_i \in 2^V$, i.e., conjunctions of positive or negative literals.

The i^{th} suffix of τ, starting at i, is denoted as τ^i, i.e., $\tau^i = \tau_i, \tau_{i+1}, \ldots, \tau_n$.

If we have two traces τ and σ of the same length, the pairwise conjunction of the literals in τ and σ is denoted as $\tau \cdot \sigma$. Further, let $W \subseteq V$. If a trace σ consist only of W-literals, we call it a W-trace.

Definition 3 (FTNL Semantics)**.** Given a finite trace $\tau = \tau_1, \tau_2, \ldots, \tau_n$, τ satisfies an FTNL formula if and only if:

$$\begin{array}{ll}
\tau^i \not\models \bot & \\
\tau^i \models p & \text{iff } p \in \tau_i \\
\tau^i \models \neg\varphi & \text{iff } \tau^i \not\models \varphi \\
\tau^i \models \varphi \vee \psi & \text{iff } \tau^i \models \varphi \text{ or } \tau^i \models \psi \\
\tau^i \models \mathbf{X}\,\varphi & \text{iff } i = n \text{ or } \tau^{i+1} \models \varphi
\end{array}$$

where $i = 1, 2, \ldots, n$.

Some formula φ is *satisfiable* iff there exists a finite trace τ such that $\tau \models \varphi$. In many cases, aside the basic operators of a logic that define its expressiveness, we use additional operators that simplify modeling and result in more intuitive specifications. These higher-level operators are often referred to as *syntactic sugar* since they do not change the expressiveness. In our case we will use also *conjunction* ($\wedge$), *implication* ($\rightarrow$), *exclusive or* ($\oplus$), and *equivalence*, based on the well-known established semantics of those operators.

Systems and Diagnoses

When employing model-based diagnosis, we use a model for reasoning about a system's nominal or faulty device behavior.

Definition 4 (System Description)**.** Given a set of variables V, a system description SD is defined as a triple

$\langle \mathrm{M}, \mathrm{COMPS}, \mathrm{OBS} \rangle$ such that M is an FTNL formula over V, COMPS is a set of special *component variables*, OBS is a set of observable variables, and $\mathrm{COMPS} \cup \mathrm{OBS} \subseteq V$.

Coming-up with a system model of the device of interest is, in general, a difficult task, especially with a low-level logic like FTNL. There are cases in which the model creation can be automated. One such case are models of Integrated Circuits (ICs). These models can be obtained from reverse-engineered images (netlists) or from languages and specifications used for synthesizing the ICs. When we are modeling an IC with state, we typically use variables instead of wires and FTNL operators to constrain the behavior of the system.

With SD we aim to capture a system's behavior such as to be able to reason about both, nominal and faulty behavior. In order to introduce *possible faults*, we use *error variables* and *fault-models*. For a class of fault models such as stuck-at-zero (S-A-0), stuck-at-one (S-A-1), behave as another component, global short-circuit, etc., the required modifications to the original model are linear (Feldman, Provan, and van Gemund 2009). (Reiter 1987) used only *abnormal* predicates without any assumptions on the faulty behavior for models in first-order logic.

By *fault augmentation* we mean the introduction of fault variables and fault constraints (modes) in the model. Consider the 2-bit counter from our running example and allow each gate's output to be stuck-at-one. We get the following FTNL model:

$$M = \left| \begin{array}{l} (\neg a_2 \rightarrow (e_2 \Leftrightarrow (e_1 \wedge o_1))) \;\wedge\; (a_2 \rightarrow e_2) \\ (\neg x_1 \rightarrow (z_1 \Leftrightarrow (e_1 \oplus o_1))) \;\wedge\; (x_1 \rightarrow z_1) \\ (\neg x_2 \rightarrow (z_2 \Leftrightarrow (e_2 \oplus o_2))) \;\wedge\; (x_2 \rightarrow z_2) \\ (\neg f_1 \rightarrow (z_1 \Leftrightarrow \mathbf{X}\, o_1)) \wedge (f_1 \rightarrow z_1) \\ (\neg f_2 \rightarrow (z_2 \Leftrightarrow \mathbf{X}\, o_2)) \wedge (f_2 \rightarrow z_2) \end{array} \right. \qquad (1)$$

Each line in Eq. 1 consists of two conjuncts modeling a single component. The first one is the healthy mode while the second mode is a stuck-at-1 fault mode. The three observable variables are $\mathrm{OBS} = \{e_1, o_1, o_2\}$ where e_1 is a primary input that enables the counter and o_1 and o_2 are the primary outputs that give the counter values. There are also five fault variables: $\mathrm{COMPS} = \{x_1, f_1, a_2, x_2, f_2\}$.

Each gate of the 2-bit counter is modeled as a top-level conjunct in Eq. 1. The flip-flops use the $\mathbf{X}$ operator. Notice that the second conjunct in each gate's model is the stuck-at-1 model. For example, the subexpression $a_2 \rightarrow e_2$ means that when gate a_2 is faulty, the wire modeled by variable e_2 assumes value $\top$ (we have $x \equiv x \Leftrightarrow \top$ for any x).

Definition 5 (Observation)**.** Given a system description SD, an observation α is defined as an OBS-trace.

The main goal of MBD is to compute diagnoses. A diagnosis is an explanation of an observation α. One can think of COMPS as a set of inputs, that are not set by the user but by the god of failures and whose values denote if there is a fault or not. If those values are hypothetical, i.e., guessed by an algorithm like the one we present in this paper, we get a diagnosis.

Definition 6 (Diagnosis)**.** Given a system description $\mathrm{SD} = \langle \mathrm{M}, \mathrm{COMPS}, \mathrm{OBS} \rangle$ and an observation α, a diagnosis ω is defined as a COMPS-trace, such that $\alpha \cdot \omega \models \mathrm{M}$.

An assignment to a fault-variable $f_3 \leftrightarrow \top$, abbreviated as f_3, means that the respective component is healthy while assigning falsity ($f_3 \leftrightarrow \perp$, abbreviated as $\neg f_3$) means that the component is faulty. In this paper the convention is to use a positive fault-polarity, i.e., the faults are positive literals. Counting the number of positive fault literals in a diagnosis ω gives the cardinality of a diagnosis. The cardinality of a diagnosis ω is denoted as $|\omega|$.

Definition 7 (Minimal-Cardinality Diagnosis)**.** A diagnosis ${}^{\leq}\omega$ is a minimal-cardinality diagnosis iff there is no other diagnosis ω' such that $|\omega'| < |{}^{\leq}\omega|$.

Our diagnostic framework consists of two parts: first, we derive a propositional formula via unrolling M, and second, we search for diagnoses with a SAT-based algorithm.

Deriving a Propositional Model

We next present an algorithm for obtaining a propositional model M_f from an FTNL formula M. The algorithm "unrolls" the FTNL formula for a time-horizon T where $T \in \mathbb{N}$.

Algorithm 1: Convert an FTNL formula to a propositional logic formula

```
Input  : M, FTNL formula, model
         $T$, integer, horizon
Output: $M_f$, propositional formula, flat model
for $t \in \{1, 2, \ldots, T\}$ do
    $M_t = \textsc{CopyIndexed}(M, t)$
    Recursively replace in $M_t$:
    (i) each $\mathbf{X}\,\varphi$ with $\varphi_{t+1}$
    (ii) each variable $v$ with $v_t$
end
$M_f = M_1 \wedge M_2 \wedge \cdots \wedge M_T$
```

Algorithm 1 simply copies the FTNL formula T times. Each iteration adds time-index to each variable and replaces the temporal operators with propositional operators that connect formulae in the current and successive time-indexed instances of the original formula. The final result is the conjunction of all time-instances of the formula. Going one instant beyond the horizon T for some internal variables is not a problem.

If we have $|V|$ variables in the original formula, the unrolled propositional formula has $T|V|$ variables. The number of clauses in the resulting formula converted to Conjunctive Normal Form (CNF) is precisely T times the number of clauses in the CNF of the original formula.

The result of applying Algorithm 1 to Equation 1 is:

$$M_f = \left| \begin{array}{l} \left[\neg x_1^1 \rightarrow \left(z_1^1 \leftrightarrow e_1^1 \oplus o_1^1\right)\right] \wedge \left[x_1^1 \rightarrow \cdots\right] \\ \left[\neg f_1^1 \rightarrow \left(o_1^2 \leftrightarrow z_1^1\right)\right] \wedge \left[f_1^1 \rightarrow \cdots\right] \\ \vdots \\ \left[\neg x_1^2 \rightarrow \left(z_1^2 \leftrightarrow e_1^2 \oplus o_1^2\right)\right] \wedge \left[x_1^2 \rightarrow \cdots\right] \\ \left[\neg f_1^2 \rightarrow \left(o_1^3 \leftrightarrow z_1^2\right)\right] \wedge \left[f_1^2 \rightarrow \cdots\right] \\ \vdots \end{array} \right. \quad (2)$$

In Equation 2, the upper index denotes the historical time-index due to the unrolling.

Theorem 1 (Soundness)**.** Let M_f be the result of applying Algorithm 2 to a system description SD. Let α_f be the flattened observation trace α, and ω_f the flattened diagnosis ω. If $\omega_f \models M_f \wedge \alpha_f$ then $\alpha \cdot \omega \models M$.

Proof (Sketch). Showing the soundness of Algorithm 2 requires an induction on the recursion depth. To show the inductive step, we have to analyze the semantics of propositional logic and FTNL.

□

While sound, Algorithm 2 is only complete for sufficiently large time horizon T. Computing T for completeness and analyzing the time horizon is a topic on its own.

Theorem 2 (Completeness)**.** Given a system description SD, an observation trace α, and a diagnosis ω, there exists a model encoding in propositional logic M_f and a propositional assignment α_f such that for any diagnostic trace ω for which $\alpha \cdot \omega \models M$ it follows that $\omega_f \models M_f \wedge \alpha_f$.

Proof (Sketch). Completeness can be shown by using induction on the time horizon T. It follows from the semantics of FTNL (Def. 3), the semantics of propositional logic, the definitions of observation and diagnosis both in FTNL and propositional logic (Feldman, Provan, and van Gemund 2010) and Alg. 1.

□

The process of unrolling of an FTNL model M leads to a linear increase in the size of M. The number of variables will be $|V| \times |T|$ and the number of propositional operators will also increase linearly.

Theorem 3 (Complexity)**.** Computing an FTNL minimal-cardinality diagnosis ω is $\Delta_3^P[O(\log n)]$-hard.

Proof. We perform a reduction from a propositional diagnostic model whose complexity is shown by Eiter and Gottlob to be in $\Delta_3^P[O(\log n)]$ (Eiter and Gottlob 1995). The reduction is $O(1)$ due to the fact that Boolean propositional models are also FTNL models. The resulting trace ω is of length one and is converted directly to a propositional diagnosis. □

Having unrolled the FTNL model, we can proceed with standard propositional MBD.

Diagnostic Algorithms

In this section we introduce a SAT-based algorithm for diagnostics. While SAT-solvers have been used extensively both in MBD (Metodi et al. 2014) and model checking, we propose a novel encoding that is more compact than state-of-the-art methods.

Standard Exhaustive Search

We brute-force the two-bit counter from Equation 2. The time horizon is $T = 5$. Fig. 2a shows the simulation output of the counter if all gates are functioning correctly. Fig. 2b show the effect of a fault in flip-flop f_1. Faults in Boolean systems propagate in a non-intuitive manner.

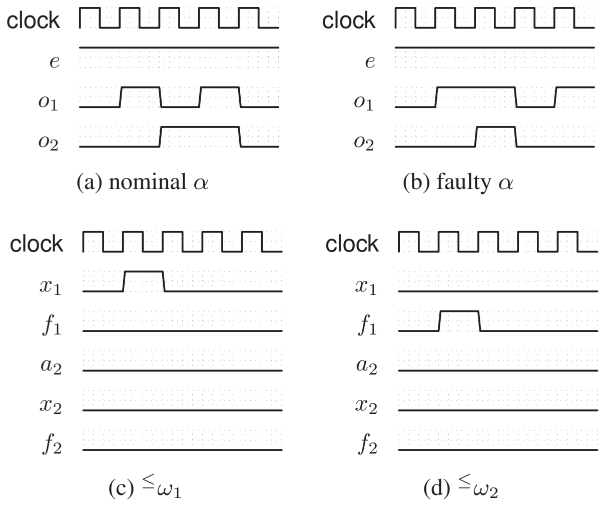


Figure 2: Observations and diagnoses of a 2-bit counter

Even the tiny running example has 25 fault variables when unrolled and the brute-force algorithm has to consider 33 554 432 possible assignments. There are two minimal-cardinality diagnoses (shown in Fig. 2c and Fig. 2d). From those, ω_1 is the injected fault and it is also discovered in the correct time. The two single-faults in ω_1 and ω_2 are indistinguishable due to the limited observability of the model. There are also 131 070 non-minimal-cardinality diagnoses.

Recall that the computational complexity of the original minimal-cardinality diagnosis problem is $\Delta_3^P[O(\log n)]$ which is in the second level of the polynomial hierarchy (Eiter and Gottlob 1995). In practice, however, this worst-case complexity is rare. Checking diagnostic assumptions on models of man-made systems such as Integrated Circuits (ICs) is typically cheap as designs are often close to planar. The number of small diagnostic assumptions is also constrained at design time by having multiple observable variables and a small number of dense subsystems.

A SAT-Based Diagnostic Algorithm

A single, not necessarily minimal, propositional diagnosis ω^f can be obtained by calling a SAT-solver as $\omega^f \models \mathrm{M}^f \wedge \alpha^f$. The process of computing a *minimal* diagnosis, however, is closer to partial MaxSAT than to SAT (Feldman et al. 2010). To achieve minimality, we add a new type of cardinality constraint (Asín et al. 2011) as illustrated in Fig. 3. In our approach, we take the original fault-augmented model and add propositional formulas that count the number of faults. We can then tie the outputs of this formula to a constant to achieve a fault-cardinality of given (minimum) size.

The augmented formula shown in Fig. 3 consists of three parts: the original propositional model $M \wedge \alpha$, a fault-counting circuit and a constant k specifying the number of faults allowed for a diagnosis. We can then initialize $k = 0$ and increase it until we find a minimal-cardinality diagnosis.

The bit-counting circuit is shown in Fig. 4. It is known as a multi-operand Boolean adder and is implemented as a ladder of multi-operand full-adders (see Fig. 4a). Each full-adder adds one bit to a binary number and consists of k half-adders where k equals the number of bits necessary for representing the binary number (see Fig. 4b). When the number of inputs is such that it is impossible to have a carry output (this is the case when the number of inputs is not a power of two), the multi-operand full-adder is implemented without a final AND-gate. The multi-operand adder uses full-adders of increasing size. The first adder has one input, the second and third have two inputs, the next four have three inputs, etc.

The number of gates in a MOA is $2^{k+1}(k-2) - k + 3$ where k is the number of outputs. The number of inputs is n where $n = 2^k - 1$ for $k \geq 2$. As a result, the MOA circuit complexity is $O(n \log n)$.

Algorithm 2: SAT-based diagnostic algorithm

```
Input  : M, propositional formula, model
         $\alpha$, propositional assignment, observation
Output: $\Omega$, set of assignment, diagnoses
$f \leftarrow M \wedge \alpha$
$S \leftarrow$ AddMultiOperandAdder($f$, COMPS)
$\Omega \leftarrow \emptyset$
$n \leftarrow 0$
while $\Omega = \emptyset$ and $n \leq |\text{COMPS}|$ do
    $\beta \leftarrow$ BinaryConstant($n$)
    $\gamma \leftarrow$ FormulaToCNF($f \wedge \beta$)
    while $\omega \leftarrow$ SolveCNF($\gamma$) do
        $\Omega \leftarrow \Omega \cup \omega$
        $\gamma \leftarrow \gamma \wedge \neg\omega$
    end
    $n \leftarrow n + 1$
end
```

Algorithm 2 shows the full-implementation of the diagnostic algorithm. The AddMultiOperandAdder appends a multi-operand adder to the original formula, calculating the number of faulty-components. The number of faults is constrained by assigning a binary-encoded number (the encoding is performed by BinaryConstant). Algorithm 2 first looks for zero-cardinality diagnoses, then for single-fault diagnoses, and so on. The algorithm terminates once minimal cardinality diagnoses are found or if n reaches the number of components in the system. In practice there is a huge number of diagnoses and we terminate the algorithm after the first k minimal cardinality diagnoses are found. The formula is converted to CNF by FormulaToCNF. The actual SAT call is done in SolveCNF.

Once a diagnostic solution has been found by the SAT-solver (SolveCNF), it is negated and blocking clauses are added to the original CNF γ. In general, this increases the difficulty of subsequent SAT-solving as the problem transits from under-constrained to constrained.

Algorithm 2 searches for diagnoses from lower-cardinality to higher-cardinality. Depending on the assumptions, though, more efficient schemes such as binary search, are possible.

Experimental Results

The algorithms described in this paper are implemented in a mixture of Python and C. For the SAT solving we have used Lingeling (Biere 2016).

In addition to the n-counter family introduced in Sec. , we have also constructed a benchmark of real-world diagnosis problems consisting of eight synchronous circuits (see Table 1). These models are translated from ISCAS-89 netlists (Brglez, Bryan, and Kozminski 1989). ISCAS-89 is a benchmark of sequential digitial circuits for testing Automated Test Pattern Generation (ATPG) algorithms. ATPG is related to MBD and many algorithms bear resemblance.

Table 1: ISCAS-89 sequential circuits

Name	Description	|OBS|	FFs	|COMPS|
s208	digital fractional multiplier	11	8	112
s298	traffic light controller	9	14	133
s349	4-bit multiplier	20	15	176
s400	traffic light controller	9	21	185
s420	digital fractional multiplier	19	16	234
s444	traffic light controller	9	21	202
s526	traffic light controller	9	21	214
s838	digital fractional multiplier	35	32	478

The circuits in Table 1 come from real-world applications. They have variable number of observable variables ($|\text{OBS}|$), flip-flops (FFs) and gates ($|\text{COMPS}|$). With the help of a simulator, we have reverse engineered their function and have manually created input signals. Several of the circuits, for example, have to be reset at start-up.

The main goal of this section is to characterize the CPU performance of Algorithm 2. To do this we measure two main metrics: (i) the time for unrolling and converting the model M to CNF (T_{CNF}) and (ii) the time for computing the first 10 diagnoses in order of cardinality (T_{SAT}) To characterize the space complexity we count the number of variables and the number of clauses in the CNF.

Figure 5 shows T_{SAT} as a function of the n-counter size n and the time-horizon T. We have experimented with $1 \leq n \leq 16$ and $1 \leq T \leq 16$. For each combination of n and T we have generated a number of diagnostic scenarios by random fault injection for a total of 31 909 scenarios.

Fig. 5 shows good scaling of T_{SAT} for the two independent variables n and T. The absolute timing is also good: the SAT solver needs only 0.22 s for computing the first minimal-cardinality diagnosis for $n = 16$ and $T = 16$. Each subsequent

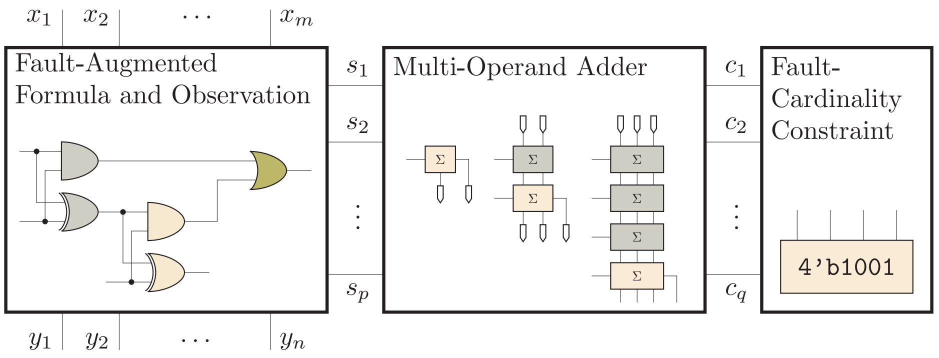


Figure 3: Augmented circuit

diagnosis of the same scenario takes longer with the tenth taking 14 s. This is due to the blocking literals.

We have compared the performance of Algorithm 2 to the fastest MBD algorithm today (Metodi et al. 2014). The main difference between our implementation and the one of Metodi et al. is in the implementation of the cardinality constraints. We use Multi-Output Adder (MOA) while Metodi et al. use Sorting Networks (SNs). The difference comes from using an unary (MOA) or binary (SN) representation for the cardinality constraints. The results are shown in Table 2.

Table 2: SAT-based performance for ISCAS-89

	T_{CNF} [s]		T_{SAT} [s]		Variables	
Name	MOA	SN	MOA	SN	MOA	SN
s208	4.3	26.4	1.1	0.3	5736	18 434
s298	5.4	37.4	2.0	0.4	6975	22 730
s349	6.2	60.7	3.2	0.7	9606	31 054
s400	6.4	69.4	3.1	0.8	10 185	33 196
s420	8.3	110.1	4.1	1.1	13 461	43 720
s444	7.0	83.3	3.5	0.9	11 304	36 895
s526	7.9	95.8	4.1	1.1	12 096	39 937
s838	20.9	463.9	24.8	4.5	30 434	99 622

Each sequential circuit in Table 2 was run with multiple random-fault scenarios with increasing number of injected faults and for $T = 3$. For each scenario we have computed up to the first 10 diagnoses in order of cardinality for a total number of 2036 scenarios and 28 205 diagnoses. The T_{CNF} and T_{SAT} columns of Table 2 show average times.

The binary constraints lead to a significantly smaller number of variables and clauses: from 19.7 to 67.6 times less variables and from 18.4 to 62 times less clauses. The resulting speed-up in converting the ISCAS-89 problems to CNF is a factor from 6.2 to 22.2. Surprisingly the SNs lead to better SAT-solver performance. The SN speed-up varies from 3.4 to 5.5. The choice of the cardinality constraints should be according to the relative RAM vs CPU cost and the number of diagnoses required in the specific use-case.

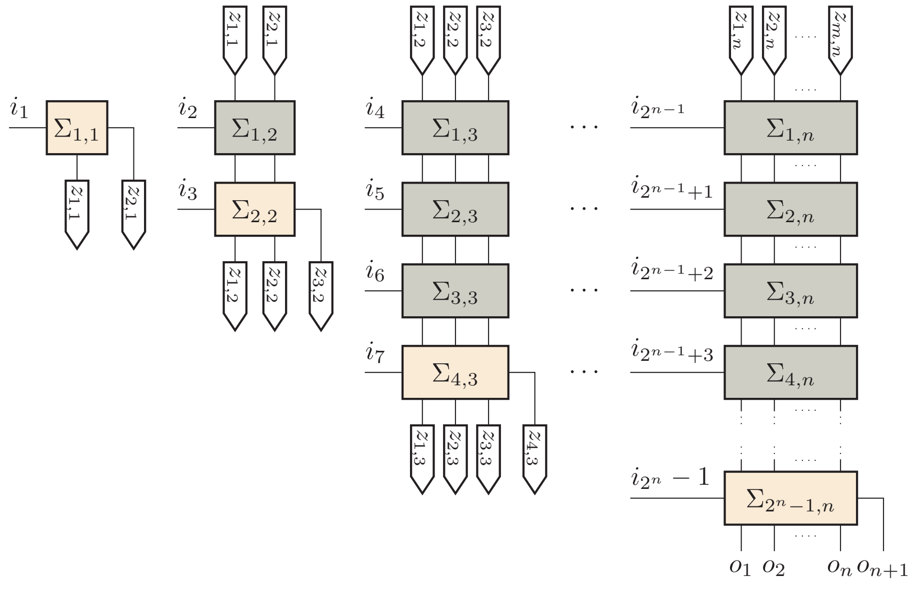
Related Work

A motivating example for our research was NASA’s Livingstone system which has been used for MBD of the Earth Observing One satellite (Hayden, Sweet, and Christa 2004). While the Livingstone framework uses automata and conflict-directed search (Williams and Nayak 1996), we provide logic characterization and a SAT-based algorithm. This results in a better understanding of the models and enables faster reasoning methods supporting larger systems.

MBD techniques similar to ours are applied to Verilog(Peischl, Riaz, and Wotawa 2012) and VHDL (Peischl and Wotawa 2006) designs and provides ISCAS-89 evaluation on single and double-faults. While, the first paper focuses on modeling, model analysis, and fault classification we provide analysis and tools for faults with higher cardinality. In the second paper, the focus is on source-level localization of VHDL faults and algorithm optimality and completeness are secondary. Another paper about applying MBD to VHDL (Friedrich, Stumptner, and Wotawa 1999) also provides some state-space reduction techniques that are othogonal to the choice of reasoning algorithms and can be applied to the algorithms presented in this paper as well.

Chen et al. used partial MaxSAT (Chen et al. 2009) for the debugging of Very-Large-Scale Integration (VLSI) designs. Our paper extends their work by experimenting with faults of multiple-cardinality as opposed to single errors. Another improvement is our introduction of the MOA encoding that considerably speeds-up the reasoning process.

Testing and formal methods are closely related to MBD. The work of (Pill and Wotawa 2018) links verification to Software-Fault Localization (SFL) (Abreu, Zoeteweij, and van Gemund 2006) by automatically creating a test oracle from requirements expressed in (F)LTL and translating (F)LTL to SAT for checking system properties. Our approach fills another gap between MBD, verification, and SFL by experimenting with circuits and offering explanations rather than checking the satisfaction of properties. Compared to diagnosing LTL specifications (Pill and Quaritsch 2013), we consider finite traces and a simpler logic aimed at representing system *implementations* like synchronous sequential circuits as opposed to logics used to describe the degrees of freedom in a specification. From a technical point of view, we furthermore do not restrict ourselves to persistent faults in a monolithic specification, but focus on all persistent and dynamic faults (including intermittent ones) as present in real systems.


(a) A ladder of half-adder-chains for multi-operand addition

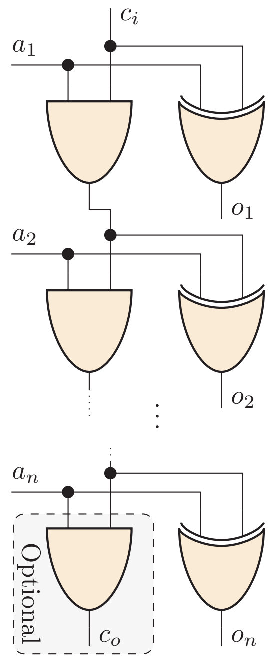

(b) A multi-operand full adder with an optional carry out bit

Figure 4: A binary multi-operand adder of variable size

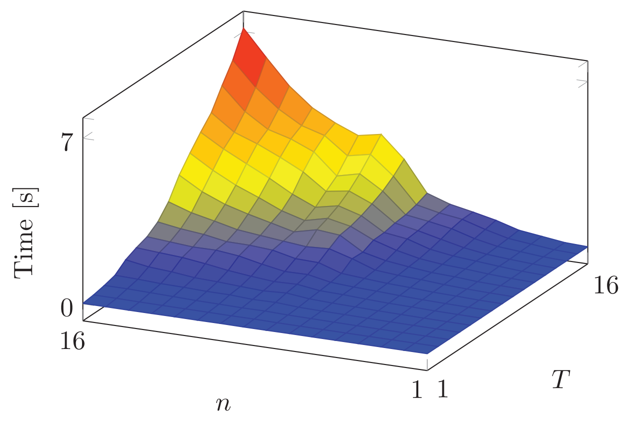


Figure 5: Average SAT time per diagnosis of the n-counter

There is a lot of literature on how to implement cardinality constraints (Nica et al. 2013). An alternative to sorting networks are cardinality networks. While they are more compact, they require putting an upper-bound on the cardinality. This will work out for systems where we limit the search to low cardinalities, but there are cases, especially in design (de Kleer, Feldman, and Matei 2018), where the fault cardinality could be equal to the number of components.

The paper of Grastien and Anbulagan is an important related-work that bridges diagnostics of Discrete Event Systems and satisfiability. Its emphasis is on the language of diagnostic queries and semantics. Our work extends this paper by analyzing and comparing encoding techniques. An interesting continuation of the two works would be to extend the benchmark proposed in this paper for the type of diagnostic questions asked in (Grastien and Anbulagan 2013).

Conclusions

Temporal logics like LTL or PSL are more expressive than propositional logic. For modeling digital ICs, for example, we only need the **X** operator though, since we are describing deterministic behavior or non-deterministic behavior where we can also explicitely count the options. We do not want full non-determinism for such systems. This restriction is the reason why we focused on a simpler variant, namely FTNL, which can be interpreted as FLTL without the until. Considering finite traces, it is possible to extend the construction to full FLTL, borrowing and adopting clauses from the FLTL test oracle construction proposed in (Pill and Wotawa 2018)

In order to be able to deal with models of real-world systems, we proposed a novel SAT-based diagnosis algorithm. Existing state-of-the-art SAT-based MBD algorithms generate many more variables and clauses. As shown for ISCAS-89 circuits, our method reduces the number of variables and clauses significantly, allowing the application of logic-based MBD to real-world-sized systems.

References

Abreu, R.; Zoeteweij, P.; and van Gemund, A. 2006. An evaluation of similarity coefficients for software fault localization. In *12th Pacific Rim Int. Symp. on Dependable Computing*, 39–46.

Abreu, R.; Zoeteweij, P.; and van Gemund, A. J. 2009. A bayesian approach to diagnose multiple intermittent faults. In *Proceedings of the Twentieth International Workshop on Principles of Diagnosis (DX'09), Stockholm Sweden*, 27–33.

Asín, R.; Nieuwenhuis, R.; Oliveras, A.; and Rodríguez-Carbonell, E. 2011. Cardinality networks: a theoretical and empirical study. *Constraints* 16(2):195–221.

Biere, A. 2016. SPLATZ, LINGELING, PLINGELING, TREENGELING, YALSAT entering the SAT competition 2016. *SAT COMPETITION 2016* 44.

Brglez, F.; Bryan, D.; and Kozminski, K. 1989. Combinational profiles of sequential benchmark circuits. In *IEEE International Symposium on Circuits and Systems*, 1929–1934.

Chen, Y.; Safarpour, S.; Veneris, A.; and Marques-Silva, J. 2009. Spatial and temporal design debug using partial MaxSAT. In *19th ACM Great Lakes symposium on VLSI*, 345–350. ACM.

Clarke, Jr., E. M.; Grumberg, O.; and Peled, D. A. 1999. *Model Checking*. Cambridge, MA, USA: MIT Press.

de Kleer, J., and Williams, B. 1987. Diagnosing multiple faults. *Artificial Intelligence* 32(1):97–130.

de Kleer, J.; Feldman, A.; and Matei, I. 2018. The duality of design and diagnosis. In *Proceedings of the Twenty-Ninth International Workshop on Principles of Diagnosis (DX'18), Warsaw, Poland*.

Eisner, C., and Fisman, D. 2006. *A Practical Introduction to PSL (Series on Integrated Circuits and Systems)*. Springer-Verlag New York, Inc.

Eiter, T., and Gottlob, G. 1995. The complexity of logic-based abduction. *Journal of the ACM (JACM)* 42(1):3–42.

Feldman, A.; Provan, G.; de Kleer, J.; Robert, S.; and van Gemund, A. 2010. Solving model-based diagnosis problems with Max-SAT solvers and vice versa. In *Proceedings of the Twenty-First International Workshop on Principles of Diagnosis (DX'10), Portland, Oregon, USA*, 185–192.

Feldman, A.; Provan, G.; and van Gemund, A. 2009. Solving strong-fault diagnostic models by model relaxation. In *21st Int.l Joint Conf. on Artificial Intelligence (IJCAI)*, 785–790.

Feldman, A.; Provan, G.; and van Gemund, A. 2010. Approximate model-based diagnosis using greedy stochastic search. *Journal of Artificial Intelligence Research* 38:371–413.

Friedrich, G.; Stumptner, M.; and Wotawa, F. 1999. Model-based diagnosis of hardware designs. *Artificial Intelligence* 111(1-2):3–39.

Grastien, A., and Anbulagan, A. 2013. Diagnosis of discrete event systems using satisfiability algorithms: A theoretical and empirical study. *IEEE Transactions on Automatic Control* 58(12):3070–3083.

Hayden, S.; Sweet, A.; and Christa, S. 2004. Livingstone model-based diagnosis of earth observing one. In *AIAA 1st Intelligent Systems Technical Conference*, 6225.

Metodi, A.; Stern, R.; Kalech, M.; and Codish, M. 2014. A novel SAT-based approach to model based diagnosis. *Journal of Artificial Intelligence Research* 51:377–411.

Nica, I.; Pill, I.; Quaritsch, T.; and Wotawa, F. 2013. The route to success - a performance comparison of diagnosis algorithms. In *23rd Int. Joint Conf. on Artificial Intelligence*, 1039–1045.

Parhami, B. 2009. *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford University Press, Inc., 2nd edition.

Peischl, B., and Wotawa, F. 2006. Automated source-level error localization in hardware designs. *IEEE Design & Test of Computers* 23(1):8–19.

Peischl, B.; Riaz, N.; and Wotawa, F. 2012. Automated debugging of Verilog designs. *International Journal of Software Engineering and Knowledge Engineering* 22(05):695–723.

Petersen, E. 2011. *Single Event Effects in Aerospace*. John Wiley & Sons.

Pill, I., and Quaritsch, T. 2013. Behavioral diagnosis of LTL specifications at operator level. In *23rd International Joint Conference on Artificial Intelligence (IJCAI)*, 1053–1059.

Pill, I., and Wotawa, F. 2018. Automated generation of (F)LTL oracles for testing and debugging. *Journal of Systems and Software* 139:124–141.

Pnueli, A. 1977. The temporal logic of programs. In *Annual Symposium on Foundations of Computer Science*, 46–57.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32(1):57–95.

Tompkins, D. A., and Hoos, H. H. 2004. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In *International conference on theory and applications of satisfiability testing*, 306–320.

Williams, B., and Nayak, P. 1996. A model-based approach to reactive self-configuring systems. In *Proceedings of the national conference on artificial intelligence*, 971–978.