Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence

Behavioral Diagnosis of LTL Specifications at Operator Level

Ingo Pill and Thomas Quaritsch
Institute for Software Technology
Graz University of Technology
Inffeldgasse 16b/II, 8010 Graz, Austria
{ipill,tquarits}@ist.tugraz.at

Abstract

Product defects and rework efforts due to flawed specifications represent major issues for a project's performance, so that there is a high motivation for providing effective means that assist designers in assessing and ensuring a specification's quality. Recent research in the context of formal specifications, e.g. on coverage and vacuity, offers important means to tackle related issues. In the currently underrepresented research direction of diagnostic reasoning on a specification, we propose a scenario-based diagnosis at a specification's operator level using weak or strong fault models. Drawing on efficient SAT encodings, we show in this paper how to achieve that effectively for specifications in LTL. Our experimental results illustrate our approach's validity and attractiveness.

1 Introduction

When a project's efficiency and the related return on investment are considered, rework efforts increasing consumed resources and time-to-market are a crucial factor. Industrial data show that about 50 percent of product defects result from flawed requirements and up to 80 percent of rework efforts can be traced back to requirement defects [Wiegers, 2001]. Surprisingly enough, traditionally, research has been focusing on using a (presumably correct) specification for design verification, and seldom aimed at assisting designers in their formulation or verifying their quality. However, and specifically for specification-driven development flows like proposed for Electronic Design Automation (EDA) by [PROSYD homepage, 2013], a specification's quality is crucial as design development/verification/synthesis depend on it.

Recently, specification development has been gaining specific attention in a formal context. Coverage and vacuity can pinpoint to specification issues [Fisman *et al.*, 2009; Kupferman, 2006], and specification development tools like IBM's RuleBase PE [RuleBasePE homepage, 2013] or the academic tool RAT [Pill *et al.*, 2006; Bloem *et al.*, 2007] help by, for example, letting a designer explore a specification's semantics. RAT's property simulation (a similar feature was designed for RuleBase PE as well [Bloem *et al.*, 2007]), for instance, lets a user explore a specification's evaluation along sample behavior (a trace). The temporal evolution for all subformulae is visualized via individual waveforms, presented according to a specification's parse tree. While industrial feedback was very good [Bloem *et al.*, 2007], this still involves manual reasoning and intervention.

We aim to increase the level of automation via diagnostic reasoning that pinpoints to specification issues more directly. RuleBase PE offers a very interesting feature in this respect, explaining counterexamples using causality [Beer *et al.*, 2009]. Reasoning about points in the trace where the prior stem's satisfiability status differs from its extension in distinctive ways, critical signals and related failure causes are identified and marked with red dots in the visualized waveform for this time-step. [Schuppan, 2012] offers unsatisfiable cores in the clauses he derives for specifications in the Linear Temporal Logic (LTL) [Pnueli, 1977].

In contrast to [Beer *et al.*, 2009] we aim at the specification rather than the trace, and on a set of diagnoses instead of a flat list of affected components. That is, accommodating Reiter's theory of diagnosis in the context of formal specifications (in LTL as a first step), we derive diagnoses that describe viable combinations of operator occurrences (subformulae) whose concurrent incorrectness explains a trace's unexpected (un-)satisfiability, both in the context of weak and strong fault modes. Our diagnoses address operator occurrences rather than clauses (a diagnosis covers all unsat cores, and we can easily have 10^6 clauses even for small specifications), so that we effectively focus the search space to the items and granularity level the designer is working with.

Adopting the interface of the established property simulation idea, we consider specific scenarios in the form of infinite lasso-shaped traces a designer can define herself or retrieve, for example, by model-checking. We use a specifically tailored structure-preserving SAT encoding for our reasoning, exploiting known trace features and weak or strong fault models. For strong fault models that include descriptions of "alternative" behavior, our diagnoses directly suggest repairs like "there should come a weak instead of a strong until".

Extending [Pill and Quaritsch, 2012], our paper is structured as follows. We cover preliminaries in Section 2, and discuss our SAT-based diagnosis approach for LTL in Section 3. Following experimental results in Section 3.1, we draw conclusions and depict future work in Section 4. Related work is discussed throughout the paper where appropriate.

2 Preliminaries

Reiter's theory of diagnosis [Reiter, 1987] defines the consistency-oriented model-based diagnosis [Reiter, 1987; de Kleer and Williams, 1987] of a system as follows: A system description SD describes the behavior of a set of interacting components $COMP$. SD contains sentences $\neg AB(c_i) \Rightarrow$ *NominalBehavior*(c_i) encoding a component's behavior under the assumption that it is not operating abnormally (the assumption predicate $AB(c_i)$ triggers a component c_i's abnormal behavior, and *NominalBehavior* defines correct behavior in first order logic). As there are no definitions regarding abnormal behavior, the approach is considered to use a *weak fault model* (WFM). Given some actually observed system behavior OBS, a system is recognized to be faulty, iff $SD \cup OBS \cup \{\neg AB(c_i) | c_i \in COMP\}$ is inconsistent. A minimal diagnosis explaining the issue is defined as follows.

Definition 1. *A* minimal diagnosis *for (SD, COMP, OBS) is a subset-minimal set $\Delta \subseteq COMP$ such that $SD \cup OBS \cup \{\neg AB(c_i) | c_i \in COMP \setminus \Delta\}$ is consistent.*

Reiter proposes to compute all the minimal diagnoses as the minimal hitting sets of the set of not necessarily minimal conflict sets for (SD, $COMP$, OBS).

Definition 2. *A* conflict set *CS for (SD, COMP, OBS) is a set $CS \subseteq COMP$ such that $SD \cup OBS \cup \{\neg AB(c_i) | c_i \in CS\}$ is inconsistent. If no proper subset of CS is a conflict set, CS is a minimal conflict set.*

Using a theorem prover capable of returning conflicts, Reiter's algorithm [Reiter, 1987] is able to obtain the sets of conflict sets and minimal diagnoses on-the-fly, where appropriate consistency checks verify a diagnosis Δ's validity, and a set of pruning rules ensures their minimality. [Greiner *et al.*, 1989] presented an improved version, HS-DAG, that uses a directed acyclic graph (DAG) instead of a tree and addresses some minor but serious flaws in Reiter's original publication.

For a *strong fault model* (SFM) approach, also abnormal behavior is defined [de Kleer and Williams, 1989], which has the advantage that diagnoses become more specific. On the downside, with n the number of assumptions and m the maximum number of modes for any assumption, the search space grows significantly from 2^n to $\mathcal{O}(m^n)$.

For our definitions of an infinite trace and the Linear Temporal Logic (LTL) [Pnueli, 1977], we assume a finite set of atomic propositions AP that induces alphabet $\Sigma = 2^{AP}$. Complementing the in- and output signals, we add further propositions for the subformulae to encode their evaluation. It will be evident from the context when we refer with a trace to the signals only. LTL is defined in the context of infinite traces, which we define as sequence as is usual. A sequence of letters with finite space (of finite length k) can describe an infinite computation only in the form of a lasso-shaped trace (with a cycle looping back from k to $0 \leq l \leq k$) as follows (or it could describe all traces that have the sequence as prefix):

Definition 3. *An infinite trace τ is an infinite sequence over letters from some alphabet Σ of the form $\tau = (\tau_0 \tau_1 \dots \tau_{l-1})(\tau_l \tau_{l+1} \dots \tau_k)^\omega$ with $l, k \in \mathbb{N}$, $l \leq k$, $\tau_i \in \Sigma$ for any $0 \leq i \leq k$, $(...)^\omega$ denoting infinite repetition of the corresponding (sub-)sequence, $(\tau_0 \tau_1 \dots \tau_{l-1})$ referring to τ's* finite stem*, l to the loop-back time step, and $(\tau_l \tau_{l+1} \dots \tau_k)$ representing the trace's* (k,l)-loop *[Biere* et al.*, 1999]. We denote the infinite suffix starting at i as τ^i, and τ_i refers to τ's element at time step i, where for any $i > k$ we have $\tau_i = \tau_{l+(i-l)\%(k-l+1)}$.*

Definition 4. *Assuming a finite set of atomic propositions AP, and δ to be an LTL formula, an LTL formula φ is defined inductively as follows [Pnueli, 1977]:*

- *for any $p \in AP$, p is an LTL formula*
- *$\neg\varphi$, $\varphi \wedge \delta$, $\varphi \vee \delta$, $\mathsf{X}\,\varphi$, and $\varphi\,\mathsf{U}\,\delta$ are LTL formulae*

Similar to τ_i and a trace, we denote with φ_i a formula's evaluation at time step i. We use the usual abbreviations $\delta \rightarrow \sigma$ and $\delta \leftrightarrow \sigma$ for $\neg\delta \vee \sigma$ and $(\delta \rightarrow \sigma) \wedge (\sigma \rightarrow \delta)$ respectively. For brevity, we will also use $\overline{p}$ to denote the negation of some atomic proposition $p \in AP$, as well as $\top / \bot$ for $p \vee \neg p / p \wedge \neg p$.

Note that the popular operators $\delta\,\mathsf{R}\,\sigma$, $\mathsf{F}\,\varphi$, $\mathsf{G}\,\varphi$, and $\delta\,\mathsf{W}\,\sigma$ not mentioned in Def. 4 are syntactic sugar for common formulae $\neg((\neg\delta)\,\mathsf{U}\,(\neg\sigma))$, $\top\,\mathsf{U}\,\varphi$, $\bot\,\mathsf{R}\,\varphi$, and $\delta\,\mathsf{U}\,\sigma \vee \mathsf{G}\,\delta$ respectively. Their semantics are thus defined only due to their use in the arbiter example depicted later on:

Definition 5. *Given a trace τ and an LTL formula φ, $\tau (=\tau^0)$ satisfies φ, denoted as $\tau \models \varphi$, under the following conditions*

$$
\begin{aligned}
&\tau^i \models p && \textit{iff } p \in \tau_i \\
&\tau^i \models \neg\varphi && \textit{iff } \tau^i \not\models \varphi \\
&\tau^i \models \delta \wedge \sigma && \textit{iff } \tau^i \models \delta \textit{ and } \tau^i \models \sigma \\
&\tau^i \models \delta \vee \sigma && \textit{iff } \tau^i \models \delta \textit{ or } \tau^i \models \sigma \\
&\tau^i \models \mathsf{X}\,\varphi && \textit{iff } \tau^{i+1} \models \varphi \\
&\tau^i \models \delta\,\mathsf{U}\,\sigma && \textit{iff } \exists j \geq i . \tau^j \models \sigma \textit{ and } \forall i \leq m < j . \tau^m \models \delta \\
&\tau^i \models \delta\,\mathsf{R}\,\sigma && \textit{iff } \forall j \geq i . \tau^j \models \sigma \textit{ or } \exists i \leq m < j . \tau^m \models \delta \\
&\tau^i \models \mathsf{F}\,\varphi && \textit{iff } \exists j \geq i . \tau^j \models \varphi \\
&\tau^i \models \mathsf{G}\,\varphi && \textit{iff } \forall j \geq i . \tau^j \models \varphi \\
&\tau^i \models \delta\,\mathsf{W}\,\sigma && \textit{iff } \tau^i \models \delta\,\mathsf{U}\,\sigma \textit{ or } \tau^i \models \mathsf{G}\,\delta
\end{aligned}
$$

3 Model-Based Diagnosis at the Operator Level for LTL Specifications

Explanations why things went wrong are always welcome when facing unexpected and surprising situations, for example, when a supposed witness contradicts a specification as in the following example taken from [Pill *et al.*, 2006]: Assume a two line arbiter with two request lines r_1 and r_2 and the corresponding grant lines g_1 and g_2. Its specification consists of the following four requirements: R_1 demanding that requests on both lines must be granted eventually, R_2 ensuring that no simultaneous grants are given, R_3 ruling out any initial grant before a request, and finally R_4 preventing additional grants until new incoming requests. Testing her specification, a designer defines a supposed witness τ (i.e. a trace that should satisfy the specification) featuring a simultaneous initial request and grant for line 1 (line 2 is omitted for clarity).

$$
\begin{aligned}
&R_1 : \forall i : \mathsf{G}\,(r_i \rightarrow \mathsf{F}\,g_i) \\
&R_2 : \mathsf{G}\neg(g_1 \wedge g_2) \qquad\qquad \tau = \begin{pmatrix} r_1 \\ g_1 \end{pmatrix}\left(\begin{matrix} \overline{r_1} \\ \overline{g_1} \end{matrix}\right)^{\omega} \\
&R_3 : \forall i : (\neg g_i \,\mathsf{U}\, r_i) \\
&R_4 : \forall i : \mathsf{G}\,(g_i \rightarrow \mathsf{X}\,(\neg g_i \,\mathsf{U}\, r_i)) \qquad (i \in \{1,2\})
\end{aligned}
$$

As pointed out by RAT's authors, using a tool like RAT, the designer would recognize that, unexpectedly, her trace contradicts the specification. However, diagnostic reasoning offering explanations *why* this is the case, would obviously be a valuable asset for debugging, which is exactly our challenge.

The specific problem in the scenario above is the until operator $\neg g_i \,\mathsf{U}\, r_i$ in R_4 that should be replaced by its *weak* version $\neg g_i \,\mathsf{W}\, r_i$: While the idea of both operators is that $\neg g_i$ should hold until r_i holds, the *weak* version does not require r_i to hold eventually, while the "strong" one does. Thus, R_4 in its current form repeatedly requires further requests that are not provided by τ, and which is presumably not in the designer's intent. Evidently, the effectiveness of a diagnostic reasoning approach depends on whether a diagnosis' impact on a specification is easy to grasp, and can pinpoint the designer intuitively to issues like the one in our example.

Before describing our diagnosis approach for LTL specifications, we now introduce our general temporal reasoning principles for LTL, which we are going to adapt accordingly.

For an LTL formula φ, we can reason about its satisfaction by a trace τ via recursively considering τ's current and next time step in the scope of φ and its subformulae. While this is obvious for the Boolean connectives and the *next* operator X, the *until* operator is more complex. The well known expansion rules as immanently present also in LTL tableaus and automata constructions [Clarke *et al.*, 1994; Somenzi and Bloem, 2000] capture this line of reasoning. We can unfold $\delta \,\mathsf{U}\, \sigma$ to $\sigma \vee \delta \wedge \mathsf{X}\,(\delta \,\mathsf{U}\, \sigma)$, which basically encodes the options of how to satisfy φ in the current time step (considering Def. 5 this is the case where $j = i$) and the option of pushing the obligation (possibly iteratively) to the next time step ($j > i$). Considering acceptance, we have to verify whether the obligation would be pushed infinitely in time. Like [Biere *et al.*, 1999] suggested for LTL model-checking, we use a SAT solver for our reasoning, with the advantage that k is known. Similar to their encoding or [Heljanko *et al.*, 2005], we encode our questions into a satisfiability problem, where, as in our case also the loop-back time step l is given, we can check for pushed obligations very efficiently in our structure-preserving variant.

In Table 1, column 2 lists our unfolding rationales that connect φ_i to the evaluations of φ's subformulae and signals in the current and next time step. A checkmark in the third column indicates whether a rationale is to be instantiated for all time steps (remember that $\tau_{k+1} = \tau_l$ due to Def. 3, so with our encoding $\varphi_{k+1} = \varphi_l$), whereas in the last column we list the corresponding clauses as added to our conjunctive normal form (CNF) encoding. In the clauses, φ_i represents the corresponding time-instantiated variable that we add for each subformula in order to derive a structure-preserving encoding. Given these clauses, we can directly obtain a SAT problem for $\tau \models \varphi$ in CNF from φ's parse tree and τ as follows.

φ	Unfolding rationales	I	Clauses
$\top/\bot$	$\varphi_i \leftrightarrow \top/\bot$	✓ (a)	$\varphi_i/\overline{\varphi}_i$
$\delta \wedge \sigma$	$\varphi_i \leftrightarrow (\delta_i \wedge \sigma_i)$	✓ (b_1)	$\overline{\varphi}_i \vee \delta_i$
		✓ (b_2)	$\overline{\varphi}_i \vee \sigma_i$
		✓ (b_3)	$\varphi_i \vee \overline{\delta}_i \vee \overline{\sigma}_i$
$\delta \vee \sigma$	$\varphi_i \leftrightarrow (\delta_i \vee \sigma_i)$	✓ (c_1)	$\varphi_i \vee \overline{\delta}_i$
		✓ (c_2)	$\varphi_i \vee \overline{\sigma}_i$
		✓ (c_3)	$\overline{\varphi}_i \vee \delta_i \vee \sigma_i$
$\neg\delta$	$\varphi_i \leftrightarrow \neg\delta_i$	✓ (d_1)	$\overline{\varphi}_i \vee \overline{\delta}_i$
		✓ (d_2)	$\varphi_i \vee \delta_i$
$\mathsf{X}\,\delta$	$\varphi_i \leftrightarrow \delta_{i+1}$	✓ (e_1)	$\overline{\varphi}_i \vee \delta_{i+1}$
		✓ (e_2)	$\varphi_i \vee \overline{\delta}_{i+1}$
$\delta \,\mathsf{U}\, \sigma$	$\varphi_i \rightarrow (\sigma_i \vee (\delta_i \wedge \varphi_{i+1}))$	✓ (f_1)	$\overline{\varphi}_i \vee \sigma_i \vee \delta_i$
		✓ (f_2)	$\overline{\varphi}_i \vee \sigma_i \vee \varphi_{i+1}$
	$\sigma_i \rightarrow \varphi_i$	✓ (g)	$\overline{\sigma}_i \vee \varphi_i$
	$\delta_i \wedge \varphi_{i+1} \rightarrow \varphi_i$	✓ (h)	$\overline{\delta}_i \vee \overline{\varphi}_{i+1} \vee \varphi_i$
	$\varphi_k \rightarrow \bigvee_{l \le i \le k} \sigma_i$	(i)	$\overline{\varphi}_k \vee \bigvee_{l \le i \le k} \sigma_i$

Table 1: Unfolding rationales and CNF clauses for LTL operators. A checkmark indicates that the clauses in the corresponding line must be instantiated over time ($0 \le i \le k$).

Definition 6. *In the context of a given infinite trace with length k and loop-back time-step l, $E_1(\psi)$ encodes an LTL formula ψ using the clauses presented in Table 1, where we instantiate for each subformula φ a new variable over time, denoted φ_i for time instance i. Please note that we assume that k and l are known inside E_1 and R.*

$$
E_1(\varphi) = \begin{cases} R(\varphi) \wedge E_1(\delta) \wedge E_1(\sigma) & \textit{for} \quad \varphi = \delta \circ_1 \sigma \\ R(\varphi) \wedge E_1(\delta) & \textit{for} \quad \varphi = \circ_2 \delta \\ R(\varphi) & \textit{else} \end{cases}
$$

with $\circ_1 \in \{\wedge, \vee, \mathsf{U}\}$, $\circ_2 \in \{\neg, \mathsf{X}\}$ and $R(\varphi)$ defined as the conjunction of the corresponding clauses in Table 1.

Definition 7. *For a given infinite trace τ (with given k), $E_2(\tau) = \bigwedge_{0 \le i \le k} \left[\bigwedge_{p_i \in \tau_i} p_i \wedge \bigwedge_{p_i \in AP \setminus \tau_i} \neg p_i\right]$ encodes the signal values as specified by τ.*

Theorem 1. *An encoding $E(\varphi, \tau) = E_1(\varphi) \wedge E_2(\tau)$ of an LTL formula φ and a trace τ as of Definitions 6 and 7 is satisfiable, $SAT(E(\varphi, \tau))$, iff $\tau \models \varphi$.*

Proof. (Sketch). The correctness regarding the Boolean operators and the temporal operator *next* ($\mathsf{X}\,(\delta)$) is trivial, so that we concentrate on the operator *until* ($\delta \,\mathsf{U}\, \sigma$).

We will start with the direction $(\tau^i \models \varphi) \rightarrow \varphi_i$: According to Def. 5, $\tau^i \models \varphi$ implies that there exists some $j \ge i$, such that $\tau^j \models \sigma$, and for the time steps $i \le m < j$ we have $\tau^m \models \delta$. Clause (g) then requires φ_j to become $\top$, and Clause (h) propagates that backward to φ_i. For the direction $\varphi_i \rightarrow (\tau^i \models \varphi)$ Clauses (f_1) and (f_2) require either the immediate satisfaction by σ_i or postpone (possibly iteratively) the occurrence of σ in time (in the latter case requiring φ_{i+1} and δ_i). According to Def. 5, the first option obviously implies $\tau^i \models \varphi$, while for the second one it is necessary to show that the obligation is not postponed infinitely such that

the existential quantifier (see Def. 5) would not be fulfilled. This is ensured by Clause (i), that, if the satisfaction of σ is postponed until k, requires there to be some σ_m, with m in the infinite k, l loop, such that $\tau^m \models \sigma$. Thus we have that φ_i implies $\tau^i \models \varphi$, and in turn $(\tau^i \models \varphi) \leftrightarrow \varphi_i$. □

Via Theorem 1, we can verify whether an infinite signal trace τ is contained in a specification φ or not. Our encoding $E(\varphi, \tau)$ forms a SAT problem in CNF that is satisfiable iff $\tau \models \varphi$. An affirmative answer is accompanied by a complete evaluation of all subformulae along the trace. For counterexamples, we obtain such an evaluation by encoding the negated specification. Via $E(\varphi, \tau)$ we can also derive (or complete) k, l witnesses (by encoding φ) and counterexamples (encoding $\neg\varphi$), due to the fact that concerning Theorem 1 we only weaken the restrictions regarding signal values for this task.

As today's SAT solvers are able to compute (minimal) unsatisfiable cores [Lynce and Silva, 2004] (MUCs) in the CNF clauses, we could implement a diagnosis approach at clause level adopting Reiter's theory. Scalability would however be a severe issue: The diagnosis space would be exponential in the number of clauses, where we can easily have 10^6 clauses for $|\tau| = |\varphi| = 200$ (see Table 3). We thus introduce for any subformula ψ of φ an assumption op_ψ encoding whether the correct operator was used for ψ. Focusing on these assumptions, the diagnosis space is exponential in the length of φ, and furthermore, we argue that the smaller diagnosis set at operator level (at which a designer works) is more intuitive.

For a WFM diagnosis we extend our encoding as follows:

Theorem 2. *Assume an updated Table 1, where each clause c is extended to $\overline{op}_\varphi \vee c$, and an assignment op to all assumptions op_ψ on φ's various subformulae ψ's correctness. An encoding $E_{WFM}(\varphi, \tau) = E_1(\varphi) \wedge E_2(\tau)$ of an LTL formula φ and a trace τ as of Definitions 6 and 7 is satisfiable, $SAT(E_{WFM}(\varphi, \tau))$, iff $\tau \models \varphi$ under assumptions op.*

For an SFM diagnosis, each operator assumption toggles between various behavioral (sub-)modes $\in \{nominal, mode_1, \ldots, mode_{n-1}\}$, where, like for the nominal one, the actual behavior has to be defined for any mode via $mode_i \Rightarrow clause$. A good example for an effective fault mode for the strong until operator (U) is suggested by our running example; use a weak until (W) instead. We extend Theorem 2 as follows, where we introduce for any subformula ψ with n modes $\mathrm{ld}(n)$ assumption bits $op_{\psi,j}$:

Theorem 3. *Assume for a formula φ with n modes $\mathrm{ld}(n)$ variables $op_{\varphi,j}$, and for a mode $0 \leq m \leq n$ the corresponding minterm $M(m)$ in these variables. Furthermore assume an updated Table 1 where each clause c describing the behavior of mode m for φ is extended to $\neg M(m) \vee c$, an assignment op to all assumptions op_ψ on φ's various subformulae ψ's modes, and a CNF formula E_3 consisting of the conjunction of all negated minterms in the operator mode variables that don't refer to a behavioral mode (for all ψ). Then, an encoding $E_{SFM}(\varphi, \tau) = E_1(\varphi) \wedge E_2(\tau) \wedge E_3$ of an LTL formula φ and a trace τ as of Definitions 6 and 7 is satisfiable, $SAT(E_{SFM}(\varphi, \tau))$, iff $\tau \models \varphi$ under assumptions op.*

The correctness of both extensions to Theorem 1 follows from that for Theorem 1 and the constructions.

$\mathsf{G}(g_1 \rightarrow \mathsf{X}(\neg g_1 \mathbf{W} r_1))$	$\mathbf{F}(g_1 \rightarrow \mathsf{X}(\neg g_1 \mathsf{U} r_1))$
$\mathbf{X}(g_1 \rightarrow \mathsf{X}(\neg g_1 \mathsf{U} r_1))$	$\mathsf{G}(g_1 \rightarrow \mathsf{X}(\neg g_1 \mathsf{U} \boldsymbol{g_2}))$
$\mathsf{G}(g_1 \rightarrow \mathsf{X}(\boldsymbol{r_1} \mathbf{R} \boldsymbol{\neg g_1}))$	$\mathsf{G}(g_1 \rightarrow \mathsf{X}(\boldsymbol{r_1} \mathsf{U} \boldsymbol{\neg g_1}))$
$\mathsf{G}(g_1 \rightarrow \mathsf{X}(\neg g_1 \mathsf{U} \boldsymbol{r_2}))$	$\mathsf{G}(g_1 \rightarrow \mathsf{X}(\boldsymbol{r_1} \mathbf{W} \boldsymbol{\neg g_1}))$
$\mathsf{G}(g_1 \rightarrow \mathbf{F}(\neg g_1 \mathsf{U} r_1))$	

Table 2: The nine SFM diagnoses for the arbiter.

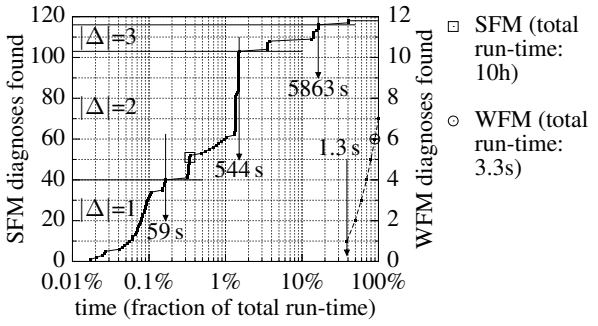
With m the number of signals, n the size of φ, o the maximum number of modes for any operator, and p the maximum number of clauses for any operator mode, upper bounds for the number of variables and clauses for $E_{\text{WFM}}(\varphi, \tau)$ and $E_{\text{SFM}}(\varphi, \tau)$ can be estimated as $\mathcal{O}((m+n)k + n \cdot \mathrm{ld}(o))$ and $\mathcal{O}((m \cdot k + n \cdot \mathrm{ld}(o) + p \cdot o \cdot k \cdot n)$ respectively. While the terms can obviously be simplified, they illustrate the origins of the variables and clauses.

Using our encodings, and varying the assumptions on the operators' correctness, we can obviously compute LTL specification diagnoses that have the desired focus on operator occurrences using various diagnosis algorithms, i.e., based on hitting conflicts like Reiter's approach or computing diagnoses directly as, e.g., suggested by [Metodi *et al.*, 2012].

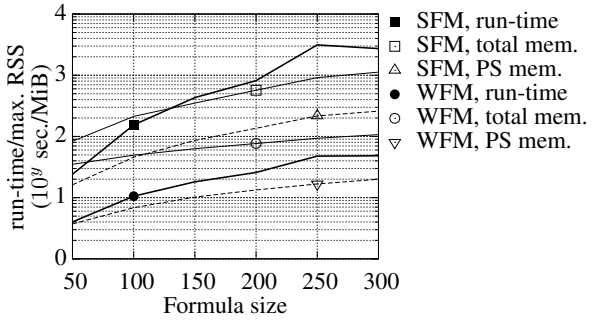
For our proof-of-concept tests, we used HS-DAG which has several interesting features for our setting. Besides being complete, due to its support of on-the-fly computations (including the conflict sets themselves), it is very efficient when limiting the desired diagnosis size. Furthermore, as diagnoses are continually found during computation, they can be reported to the user instantly, which is an attractive feature for interactive tools. For SFM models, we made HS-DAG aware of strong fault modes adopting a notion of conflicts similar to [Nyberg, 2011]. We implemented basic fault models like confusion of Boolean operators, confusion of unary temporal operators, confusion of binary temporal operators, twisted operands for binary temporal operators and "use variable v_j instead of v_i". Using a SAT solver that is able to return MUCs, we extract those unit clauses from the returned core that assign operator assumptions. Those sets represent our (not necessarily minimal) conflict sets on the operators.

For the arbiter, our WFM approach results in five single fault diagnoses. All five concern (faulty) R_4 for line 1, and when rewriting the implication, they are the bold-faced operators in $\mathbf{G}(\boldsymbol{\neg} g_1 \boldsymbol{\vee} \mathbf{X}(\neg g_i \mathbf{U} r_i))$. Intuitively, when considering the parse tree, those diagnoses farthest from the root should be prioritized during debugging, which would be the incorrect *until* for our arbiter example. Our SFM approach derived the nine diagnoses listed in Table 2, where the first one "replace $\neg g_1 \mathsf{U} r_1$ by $\neg g_1 \mathsf{W} r_1$" catches the actual fault.

While we considered operators F, G, W, and R as (easily rewritable) syntactic sugar for the Theorems, we actually translate these operators directly, reducing the number of variables and clauses, and without rewriting, our diagnoses directly address the operators as originally specified by the designer. Due to lack of space we do not report the corresponding clauses (see [Pill and Quaritsch, 2012]), which however can be easily constructed (with further optimization potential) from those for the until operator via the correspond-



(a) Number of diagnoses found over time for a sample with $|\varphi| = 100$ and $|\tau| = 200$.



(b) Run-time and memory scalability.

Figure 1: Diagnosis performance for random samples.

ing rewrite rules. For the weak until operator, e.g., clause (i) is replaced by $\varphi_k \vee \bigvee_{l \le i \le k} \overline{\delta}_i$.

Sharing/reusing subformulae in an encoding (for example if the subformula $a \,\mathsf{U}\, b$ occurs twice in a specification) could save variables and entail speedups for satisfiability checks. However, this would be counterproductive in a diagnostic context due to situations when only one instance is at fault.

Similarly to instrumenting the specification, we could take the specification as granted (with no instrumenting assumptions) and ask what is wrong with the trace. Via filtering those unit clauses in $E_2(\tau)$ from the unsat core defining the signals in τ, there is even no need for any instrumentation when considering the *weak* fault model (which is a strong model in the sense that then the value should be flipped).

3.1 Experimental Results

While the exemplary diagnoses for the arbiter illustrate the viability of our approach, in the following we discuss results for larger samples. We implemented our specification diagnosis approach using Python (CPython 2.7.1) for both the encoding and the HS-DAG algorithm. As SAT solver, we used the most recent minimal unsat core-capable PicoSAT [Biere, 2008] version 936. We ran our tests on an early 2011-generation MacBook Pro (Intel Core i5 2.3GHz, 4GiB RAM, SSD) with an up-to-date version of Mac OS X 10.6, the GUI and swapping disabled, and using a RAM-drive for the file system.

As already mentioned, the on-the-fly nature of HS-DAG allows us to report diagnoses to the user as soon as their validity has been verified. In this context, we investigated the temporal distribution of solutions for both WFM and SFM diagnosis runs. Figure 1(a) shows the number of diagnoses found for any fraction of the total computation time of a single run with a random formula of length 100, derived as suggested in [Daniele *et al.*, 1999] with $N = \lfloor |\varphi|/3 \rfloor$ variables and a uniform distribution of LTL operators. We introduced a single fault in order to derive φ_m from φ, and using our encoding we derived an assignment for $\tau \wedge \varphi \wedge \neg\varphi_m$ that defines τ for $k = 200$ and $l = 100$. We then solved the diagnosis problem $E(\varphi_m, \tau)$. For a WFM, the computation finished in 3.3 seconds discovering 7 single fault diagnoses (note that the x-axis is given in logarithmic scale). For an SFM, we stopped HS-DAG after 10 hours, with 118 diagnoses computed with a memory footprint of 1 GiB. It is important to note, however, that within one minute, all 40 single fault diagnoses were found, more than can be presumably investigated by a user in this time. All the 63 double fault diagnoses were identified after 9 minutes, and thus the majority of the 10 hours were spent for another 15 diagnoses with cardinality 3 or 4.

		run-time (sec)		RSS (MiB)	SAT	DAG	
ID	$\lvert AP\rvert$	encoding *SAT*	**total**	**total** *SAT*	#clauses *#vars*	#nodes	#Δ
1	36	11.62 *1073.36*	**1168.72**	**531.91** *131.84*	1274676 *47894*	813	262
2	39	12.81 *1344.26*	**1465.12**	**579.93** *139.79*	1382874 *48500*	939	342
3	37	9.08 *387.57*	**429.01**	**526.46** *130.46*	1260855 *48076*	296	126
1	36	0.46 *6.51*	**36.12**	**76.16** *13.50*	83980 *34737*	56	52
2	39	0.47 *7.84*	**43.67**	**77.62** *13.49*	83587 *34935*	68	61
3	37	0.45 *3.56*	**19.26**	**77.07** *13.69*	84383 *35339*	28	27

Table 3: Run-time, memory and SAT statistics for 3 samples ($|\varphi| = |\tau| = 200$) and SFM (top) as well as WFM (bottom).

In Figure 1(b) we show some results regarding diagnosis scalability for random samples with varying sizes. For any $|\varphi|$ in $\{50, 100, \ldots, 300\}$ we generated 10 random formulae as above, and restricted the search to single fault diagnoses. Restricting the cardinality of desired solutions is common practice in model-based diagnosis, and as can be seen from the last column in Table 3, we still get a considerable amount of diagnoses. In Figure 1(b), we report the average total run-time, as well as the maximum resident set size (RSS) of our whole approach and the part for PicoSAT (PS). The performance using a WFM is very attractive, with average run-times below 50 seconds and a memory footprint of approx. 100 MiB (1 MiB $= 2^{20}$ bytes vs. 1 MB $= 10^6$ bytes) even for samples with $|\varphi| = 300$. As expected, the performance disadvantage for SFM against WFM is huge; up to two orders of magnitude for the run-time and up to one for memory (maximum resident size). Identifying an effective mode-set, and tool options to focus the diagnosis on certain operators/subformulae (avoiding the instrumentation of all others) seem crucial steps to retain resources for large samples.

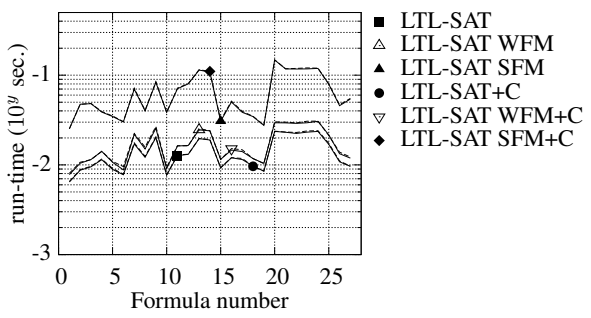


Figure 2: Run-times for a test set containing 27 common formulae taken from [Somenzi and Bloem, 2000].

$\|\varphi\|$	$\|\tau\|$	LTL-SAT			LTL-SAT WFM			LTL-SAT SFM		
		r-t.	#V	#C	r-t.	#V	#C	r-t.	#V	#C
10	10	.006	109	247	.007	118	255	0.014	178	1144
	50	.012	640	1317	.013	650	1325	0.043	949	6145
	100	.016	920	2255	.018	927	2262	0.070	1349	10516
50	10	.015	922	1386	.017	1003	1430	0.076	1554	8984
	50	.044	4365	6978	.048	4442	7023	0.311	6385	45425
	100	.082	8840	14012	.092	8918	14058	0.602	12659	89950
100	10	.025	1995	2785	.029	2175	2877	0.202	3507	23949
	50	.086	12080	13982	.097	12302	14075	0.819	17553	116395
	100	.172	30400	29045	.185	30685	29142	1.630	43901	234125

Table 4: Rounded # of variables(V) / clauses(C) and run-time for varying $|\varphi|$, $|\tau|$ (averaged over 100 traces/10 formulae).

Table 3 offers run-time (total and those parts for the SAT solver and creating the encoding), memory and encoding details for WFM and SFM single fault diagnosis of three samples with $|\varphi| = |\tau| = 200$. Apparently the majority of computation time is spent in the SAT solver tackling a multitude of encoding instances, and most of the memory footprint is related to the diagnosis part (the DAG). Thus we report also results for solving a single encoding instance in the following.

Figure 2 shows the run-times for a set of 27 formulae taken from [Somenzi and Bloem, 2000] for 100 random traces with $k = 100$ and $l = 50$. These traces were generated such that each signal is true at a certain time step with a probability of 0.5. The graph compares different variants of our encoding and the effects of enabling MUC computation, where Table 4 reports variable and clause numbers when scaling $|\varphi|$ and $|\tau|$.

Compared to an uninstrumented encoding, a WFM one is only slightly slower and an SFM variant experiences a penalty of about factor 5. Apparently, the computation of unsat cores is very cheap, making the "+C" lines nearly indistinguishable from the standard ones. Lacking the space to report corresponding numbers, we experimented also with other SAT solvers (MiniSAT 2.2 [Eén and Sörensson, 2003] and Z3 4.1 [de Moura and Bjørner, 2008]), comparing LTL-SAT to encoding LTL semantics as an SMT problem with quantifiers over uninterpreted functions in Z3. Regardless of the SAT-solver, LTL-SAT outperformed the "naïve" SMT approach, with Z3 in the lead. Presumably due to the returned cores, Z3 proved to be significantly slower than PicoSAT in the diagnosis case, so that we chose the latter for our diagnosis runs.

4 Summary and Future Work

In this paper, we proposed a novel diagnostic reasoning approach that assists designers in tackling LTL specification development situations where, unexpectedly, a presumed witness fails or a presumed counterexample satisfies a given formal specification. For such scenarios, we provide designers with complete (with respect to the model) sets of diagnoses explaining possible issues. Using the computationally cheaper weak fault model (there are no obligations on faulty operators), a diagnosis defines (a set of) operator occurrences, whose simultaneous incorrectness explains the issue. Defining also abnormal behavior variants in a strong fault model makes computation harder, but diagnoses become more precise in delivering also specific repairs (e.g., "for that occurrence of the *release* operator flip the operands").

Our implementation for the Linear Temporal Logic, which is a core of more elaborate industrial-strength logics such as PSL and is used also outside EDA, e.g. in the context of Service Oriented Architectures [García-Fanjul *et al.*, 2006], showed the viability of our approach. In contrast to Schuppan's approach [Schuppan, 2012], a designer can define scenarios and ask concrete questions (via the trace), and is supplied with (multi-fault) diagnoses addressing a specification's operator occurrences, rather than unsatisfiable cores of derived clauses. Compared to RuleBase PE's trace explanations via causality reasoning [Beer *et al.*, 2009], we address the specification rather than the trace and provide more detail compared to the set of "red dots" on the trace.

Based on Reiter's diagnosis theory, we use a structure-preserving SAT encoding for our reasoning about a presumed witness' or counterexample's relation to a specification, exploiting the knowledge about a trace's description length k and loop-back time step l. While we used HS-DAG for our tests, our WFM or SFM enhanced encoding obviously supports also newer algorithms like [Stern *et al.*, 2012], and can also be used in approaches computing diagnoses directly [Metodi *et al.*, 2012].

Extending our implementation optimizations, the latter also suggests directions for future encoding optimizations, like transferring the concept of dominating gates to specifications. Exploring incremental SAT approaches [Shtrichman, 2001] will also provide interesting results.

Compared with synthesizing a fix for an operator by solving a game, (our) individual strong fault model variants precisely define the corresponding search space. While this is good in a computational sense, it could lead to missed options, where an elaborate evaluation of fault mode-sets will be interesting future work.

Future research will aim also at accommodating multiple traces in a single diagnosis DAG, while our main objective is supporting constructs from more elaborate languages, like regular expressions (SEREs in PSL) and related operators, i.e., suffix implication and conjunction.

Acknowledgments

Our work has been funded by the Austrian Science Fund (FWF): P22959-N23. We would like to thank the reviewers for their comments and Franz Wotawa for fruitful discussions.

References

[Beer *et al.*, 2009] I. Beer, S. Ben-David, H. Chockler, A. Orni, and R. Trefler. Explaining counterexamples using causality. In *Computer Aided Verification*, pages 94–108, 2009.

[Biere *et al.*, 1999] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 193–207, 1999.

[Biere, 2008] A. Biere. PicoSAT essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:75–97, 2008.

[Bloem *et al.*, 2007] R. Bloem, R. Cavada, I. Pill, M. Roveri, and A. Tchaltsev. RAT: A tool for the formal analysis of requirements. In *Computer Aided Verification*, pages 263–267, 2007.

[Clarke *et al.*, 1994] E. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. In *Formal Methods in System Design*, pages 415–427, 1994.

[Daniele *et al.*, 1999] M. Daniele, F. Giunchiglia, and M. Vardi. Improved automata generation for Linear Temporal Logic. In *Computer Aided Verification*, pages 681–681, 1999.

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

[de Kleer and Williams, 1989] J. de Kleer and B. C. Williams. Diagnosis with behavioral modes. In *International Joint Conference on Artificial Intelligence*, pages 1324–1330, 1989.

[de Moura and Bjørner, 2008] L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, 2008.

[Eén and Sörensson, 2003] N. Eén and N. Sörensson. Minisat v1.13–a SAT solver with conflict-clause minimization. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 502–518, 2003.

[Fisman *et al.*, 2009] D. Fisman, O. Kupferman, S. Sheinvald-Faragy, and M.Y. Vardi. A framework for inherent vacuity. In *International Haifa Verification Conference on Hardware and Software: Verification and Testing*, pages 7–22, 2009.

[García-Fanjul *et al.*, 2006] J. García-Fanjul, J. Tuya, and C. de la Riva. Generating test cases specifications for compositions of web services. In *International Workshop on Web Services Modeling and Testing*, pages 83–94, 2006.

[Greiner *et al.*, 1989] R. Greiner, B. A. Smith, and R. W. Wilkerson. A correction to the algorithm in Reiter's theory of diagnosis. *Artificial Intelligence*, 41(1):79–88, 1989.

[Heljanko *et al.*, 2005] K. Heljanko, T. Junttila, and T. Latvala. Incremental and complete bounded model checking for full PLTL. In *Computer Aided Verification*, pages 98–111, 2005.

[Kupferman, 2006] O. Kupferman. Sanity checks in formal verification. In *International Conference on Concurrency Theory*, pages 37–51, 2006.

[Lynce and Silva, 2004] I. Lynce and J. P. Marques Silva. On computing minimum unsatisfiable cores. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 305–310, 2004.

[Metodi *et al.*, 2012] A. Metodi, R. Stern, M. Kalech, and M. Codish. Compiling model-based diagnosis to Boolean satisfaction. In *AAAI Conference on Artificial Intelligence*, pages 793–799, 2012.

[Nyberg, 2011] M. Nyberg. A generalized minimal hitting-set algorithm to handle diagnosis with behavioral modes. *IEEE Transactions on Systems, Man and Cybernetics–Part A: Systems and Humans*, 41(1):137–148, 2011.

[Pill and Quaritsch, 2012] I. Pill and T. Quaritsch. An LTL SAT encoding for behavioral diagnosis. In *International Workshop on the Principles of Diagnosis*, pages 67–74, 2012.

[Pill *et al.*, 2006] I. Pill, S. Semprini, R. Cavada, M. Roveri, R. Bloem, and A. Cimatti. Formal analysis of hardware requirements. In *Conference on Design Automation*, pages 821–826, 2006.

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

[PROSYD homepage, 2013] PROSYD homepage, 2013. http://www.prosyd.org.

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

[RuleBasePE homepage, 2013] RuleBasePE homepage, 2013. https://www.research.ibm.com/haifa/projects/verification/RB_Homepage.

[Schuppan, 2012] V. Schuppan. Towards a notion of unsatisfiable and unrealizable cores for LTL. *Science of Computer Programming*, 77(7–8):908–939, 2012.

[Shtrichman, 2001] O. Shtrichman. Pruning techniques for the SAT-based bounded model checking problem. In *Correct Hardware Design and Verification Methods*, pages 58–70, 2001.

[Somenzi and Bloem, 2000] F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *Conference on Computer Aided Verification*, pages 248–263, 2000.

[Stern *et al.*, 2012] R. Stern, M. Kalech, A. Feldman, and G. Provan. Exploring the duality in conflict-directed model-based diagnosis. In *AAAI Conference on Artificial Intelligence*, pages 828–834, 2012.

[Wiegers, 2001] K. E. Wiegers. Inspecting requirements. *StickyMinds Weekly Column*, July 2001. http://www.stickyminds.com.