# An LTL SAT Encoding for Behavioral Diagnosis

**Ingo Pill**[1], **Thomas Quaritsch**[1]

[1] *Institute for Software Technology, Graz University of Technology, Inffeldgasse 16b/II, Graz, 8010, Austria*
*{ipill,quaritsch}@ist.tugraz.at*

## ABSTRACT

Assisting designers in writing high-quality specifications is an important step towards minimizing product defects and rework efforts. Drawing on the attractive performance of satisfiability solvers, in this paper, we present a SAT encoding that enables an efficient model-based diagnosis of LTL specifications in the context of behavioral samples (traces). The resulting diagnoses at operator level and our experimental results illustrate our approach's viability and attractiveness.

## 1 INTRODUCTION

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 verifying designs against their specifications, and seldom aimed at verifying a specification's quality or assisting designers in their formulation. Recently, specification development has been gaining attention in a formal context. Coverage and vacuity can pinpoint to specification issues (Fisman *et al.*, 2009; Kupferman, 2006), and recent work regarding unsatisfiable cores for Linear Temporal Logic (LTL) specifications (Schuppan, 2012) provides hints at contradicting clauses in a specification's model. Furthermore, formal specification development tools like RAT (Pill *et al.*, 2006; Bloem *et al.*, 2007a) help designers in understanding formal semantics via a user-friendly interface. They allow a user, for example, to explore a specification's semantics in the scope of behavioral examples (i.e. traces, a common concept known well enough), where a user can also ask concrete questions. This way the designer can grasp the intrinsics of the specification language, and by manual inspection of its behavioral aspects she gains confidence in (and a deep understanding of) developed specifications. Diagnostic reasoning in case things are not as expected could enable further automation and would help in enhancing user experience. In this context, let us consider the following development step example for an arbiter, taken from (Pill *et al.*, 2006). The two-line arbiter's in- and outputs consist of two request lines $r_1$ and $r_2$ and the corresponding grant lines $g_1$ and $g_2$. At a certain step, the designer established the following four requirements (see Section 2 for an introduction to LTL):

$R_1$: requests on both lines must be granted eventually
$\forall i \in \{1,2\} : \mathsf{G}\,(r_i \rightarrow \mathsf{F}\,g_i)$

$R_2$: no simultaneous grants for lines 1 and 2
$\mathsf{G}\,\neg(g_1 \wedge g_2)$

$R_3$: no initial grant before a request
$\forall i \in \{1,2\} : (\neg g_i\,\mathsf{U}\,r_i)$

$R_4$: no additional grants until there is a new request
$\forall i \in \{1,2\} : \mathsf{G}\,(g_i \rightarrow \mathsf{X}\,(\neg g_i\,\mathsf{U}\,r_i))$

In order to check the specification, the designer comes up with the following supposed witness (i.e. a trace $\tau$ that should be included in the specification) featuring a simultaneous initial request and grant for line 1:

$$\tau = \begin{pmatrix} r_1, \neg r_1 \\ g_1, \neg g_1 \end{pmatrix} \begin{pmatrix} \neg r_1 \\ \neg g_1 \end{pmatrix}^{\omega}$$

Please note that for clarity of the running example, we omit the correct trace parts for line 2, while, as supposed, the designer and our approach would consider the whole trace. Using a requirements tool like RAT, the designer would recognize that, unexpectedly, her trace is not featured by the specification. Diagnostic reasoning offering explanations *why* this is the case (in the example, the operator $\mathsf{U}$ in $R_4$ should be replaced by its weak cousin $\mathsf{W}$) would be a valuable asset in this situation.

To this purpose, we propose in this paper an efficient SAT encoding for LTL in the scope of infinite traces, and use it as basis for diagnosing issues at specification and trace level. For our encoding we temporally unfold a specification's obligations to be satisfied by a trace, where similar to the encoding Schuppan uses in (Schuppan, 2012) or that of Heljanko et al. in (Heljanko *et al.*, 2005), the rationale behind the unrolling are the well known expansion rules as present in LTL tableaus and automata constructions (e.g. (Clarke *et al.*, 1994; Somenzi and Bloem, 2000)). In our specific scenario of a given infinite behavior (a trace in the form of a $k, l$-loop (Biere *et al.*, 1999) – see Section 2), we can efficiently encode the obligations pushed in time that are usually captured by more general and complex acceptance conditions. We implement a structure-preserving encoding that adds a variable for each subformula's evaluation, which enables us to efficiently instrument the encoding (and add fault models) for a model-based diagnosis with weak and strong fault modes at specification operator level. That is, in contrast to providing incompatible clause sets of a specification's derived model (Schuppan, 2012), we offer diagnoses at operator level (i.e., for the running example we pinpoint to the *until* operator ($\mathsf{U}$) in $R_4$). Aside that, the evaluation of all subformulae as presented in tools like RAT is automatically offered in the assignment reported by the SAT solver. We derive our encoding (in conjunctive normal form) directly from locally considering the single operators in the syntax tree.

Our paper is structured as follows. The preliminaries are covered in Section 2. We present the details of our basic encoding in Section 3, where we also assess its correctness. A corresponding performance evaluation is reported in Section 4. In Section 5, we show how to adapt our basic encoding for model-based diagnosis. We draw our conclusions as well as discuss related and future work in Section 6.

## 2 PRELIMINARIES

For our definitions of a trace and the Linear Temporal Logic (LTL) (Pnueli, 1977), we assume a finite set of atomic propositions $AP$ that induces the alphabet $\Sigma = 2^{AP}$. Please note that in our context $AP$ might not only cover the in- and output signals, but we could add a proposition for any subformula.

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

**Definition 2.** *Assuming a finite set of atomic propositions $AP$, and $\delta$ to be an LTL formula, an LTL formula $\varphi$ 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 $\tau_i$ and a trace, we denote with $\varphi_i$ 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. 2 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, so that their temporal meaning is given only for illustration in the following definition.

The satisfaction of an LTL formula $\varphi$ by a trace $\tau$ is defined as follows.

**Definition 3.** *Given a trace $\tau$ and an LTL formula $\varphi$, $\tau(=\tau^0)$ satisfies $\varphi$, denoted as $\tau \models \varphi$, under the following conditions*

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

Given a system description $SD$, a set of components $COMP$, as well as observations $OBS$, we peruse Reiter's definitions of a minimal diagnosis and a (minimal) conflict set (Reiter, 1987), where $AB(c)$ encodes that component $c$ is working abnormally:

**Definition 4.** *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 the minimal diagnoses via the minimal hitting sets of the set of (minimal) conflict sets for ($SD$, $COMP$, $OBS$), where a conflict set is defined as follows:

**Definition 5.** *A (minimal)* conflict set *$CS$ for (SD, COMP, OBS) is a (subset-minimal) set $CS \subseteq COMP$ such that $SD \cup OBS \cup \{\neg AB(c_i)|c_i \in CS\}$ is inconsistent.*

Reiter's algorithm (Reiter, 1987; Greiner *et al.*, 1989) which we refer to as HS-DAG, is able to obtain the set of conflict sets on-the-fly, where appropriate consistency checks verify a diagnosis $\Delta$'s validity, and a set of pruning rules ensures their minimality.

## 3 A SAT ENCODING FOR LTL IN THE CONTEXT OF INFINITE TRACES WITH GIVEN BOUNDS

An LTL formula $\varphi$ allows us to decide its satisfaction by a trace $\tau^i$ via (recursively) considering the current and next time step in the scope of $\varphi$ and its subformulae. This local consideration option is captured by the well-known LTL expansion rules as present in some form in many tableaus and automata constructions (Clarke *et al.*, 1994; Somenzi and Bloem, 2000). While the expansion is obvious for the temporal *next* operator $\mathsf{X}\,(\varphi)$, the *until* $\varphi\,\mathsf{U}\,\delta$ can be expanded to $\delta \vee \varphi \wedge \mathsf{X}\,(\varphi\,\mathsf{U}\,\delta)$, making the options for current and future satisfaction (and the corresponding obligations) more obvious. That the obligations are finally met and

Table 1: Unfolding rationale and CNF clauses for LTL operators. A checkmark indicates that the clauses in the corresponding line must be instantiated over time ($0 \leq i \leq k$, where according to Def. 1, $k+1=l$).

| $\varphi$ | Unfolding rationale | 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) $\overline{\delta}_i \vee \overline{\varphi}_{i+1} \vee \varphi_i$ | | |
| | $\varphi_k \rightarrow \bigvee_{l \leq i \leq k} \sigma_i$ | | (i) $\overline{\varphi}_k \vee \bigvee_{l \leq i \leq k} \sigma_i$ | | |

are not infinitely pushed is usually ensured by the acceptance conditions of derived automata (Clarke *et al.*, 1994; Somenzi and Bloem, 2000). As suggested in the introduction, we also implement the concept of a temporal unfolding, and as we assume the bounds $k$ and $l$ of a trace to be given, we can efficiently catch obligations pushed in time (see, e.g., Clause (i) in Table 1), instead of encoding the usual, complex acceptance conditions associated with automata constructions. Table 1 lists in column 2 our unfolding rationale that connect $\varphi_i$ to the evaluations of $\varphi$'s subformulae (including signals) in the current and next time step. While 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. 1), we show in the last column the corresponding clauses (disjunctions of possibly negated literals in $AP$) as added to the conjunctive normal form (CNF) encoding. Adding for each subformula in the parse tree a corresponding variable to $AP$, in the last column $\varphi_i$ stands for the corresponding time-instantiated variable used to connect the various operators (a structure-preserving encoding). Collecting the corresponding clauses, we can directly obtain a SAT problem in CNF from $\varphi$'s parse tree and the signals' trace as follows.

**Definition 6.** *In the context of a given infinite trace with length k and loop-back time-step l, $E_1(\varphi)$ encodes an LTL formula $\varphi$ using the clauses presented in Table 1, where we instantiate for each subformula $\varphi$ a new variable over time, denoted $\varphi_i$ 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 $\tau$ (with given k), $E_2(\tau) = \bigwedge_{0 \leq i \leq k} \left[\bigwedge_{p \in \tau_i} p \wedge \bigwedge_{p \in AP \setminus \tau_i} \neg p\right]$ encodes the signals' obligations as specified by $\tau$.*

**Theorem 1.** *For an infinite trace $\tau$, an encoding $E(\varphi,\tau) = E_1(\varphi) \wedge E_2(\tau)$ of the LTL formula $\varphi$ and the trace $\tau$ 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 more elaborate temporal operator *until* $(\delta\,\mathsf{U}\,\sigma)$.

We will start with the direction $(\tau^i \models \varphi) \rightarrow \varphi_i$: According to Def. 3, $\tau^i \models \varphi$ implies that there is some $j \geq i$, such that $\tau^j \models \sigma$ and for the time steps $i \leq m < j$ we have $\tau^m \models \delta$. Clause (g) requires $\varphi_j$ to become $\top$, and Clause (h) propagates that requirement backward to $\varphi_i$. $\varphi_i \rightarrow (\tau^i \models \varphi)$: Clauses ($f_1$) and ($f_2$) require either the immediate satisfaction by $\sigma_i$ or postpone (possibly iteratively) the occurrence of $\sigma$ in time, requiring $\varphi_{i+1}$ and $\delta_i$ in the latter case. According to Def. 3, the first option obviously implies $\tau^i \models \varphi$, while for the second one it is necessary to show that the obligation is not infinitely postponed such that the existential quantifier (see Def. 3) would not be fulfilled. This is ensured by Clause (i), that, if the satisfaction of $\sigma$ is postponed until $k$, requires there to be some $\sigma_m$, with $m$ in the infinite $k,l$ loop, such that $\tau^m \models \sigma$ (restricting the postponing). Thus we have that $\varphi_i$ implies $\tau^i \models \varphi$, and in turn $(\tau^i \models \varphi) \leftrightarrow \varphi_i$. □

Using Theorem 1, we can verify whether an infinite trace $\tau$ of the signals' values (an LTL formula does not discriminate between in- and outputs per se) is contained in a specification $\varphi$ or not. Our encoding $E(\varphi,\tau)$ forms a SAT problem already in CNF to be directly tackled by a SAT-solver, without the need of an intermediate CNF-conversion step.

If a trace $\tau$ is included in the specification $\varphi$ (i.e., $\tau \models \varphi$), the problem $E(\varphi,\tau)$ is satisfiable and the SAT solver's returned assignment contains also the evaluation of all subformulae. In the unsatisfiable case ($\tau \not\models \varphi$), we get UNSAT, and if desired and the engine is capable of, a *minimal unsatisfiable core* (Liffiton and Sakallah, 2005) more or less directly hinting at issues (those of an instrumented encoding are exploited in Section 5). The subformulae's evaluation can then be derived by solving $E(\neg\varphi,\tau)$, where a lot of clauses can be reused for the encoding. Please note that in some applications it may be reasonable to mark signals at some time steps as "do not care" by leaving them undefined. Intuitively, thus our encoding can also be used to derive (or complete) a $k,l$ witness (by encoding $\varphi$) or counterexample (by encoding $\neg\varphi$) for a specification $\varphi$. The validity of this follows directly

Table 2: Unfolding rationale and CNF clauses for syntactic sugar (*... only instantiate for $0 \leq i < l$).

| $\varphi$ | Unfolding rationale | I | Clauses |
|---|---|---|---|
| $\delta\,\mathsf{W}\,\sigma$ | $\varphi_i \rightarrow$ | ✓ | (j₁) $\overline{\varphi}_i \vee \sigma_i \vee \delta_i$ |
| | $(\sigma_i \vee (\delta_i \wedge \varphi_{i+1}))$ | | (j₂) $\overline{\varphi}_i \vee \sigma_i \vee \varphi_{i+1}$ |
| | $\sigma_i \rightarrow \varphi_i$ | ✓ | (k) $\overline{\sigma}_i \vee \varphi_i$ |
| | $\delta_i \wedge \varphi_{i+1} \rightarrow \varphi_i$ | ✓ | (l) $\overline{\delta}_i \vee \overline{\varphi}_{i+1} \vee \varphi_i$ |
| | $\bigwedge_{l \leq i \leq k} \delta_i \rightarrow \varphi_k$ | | (m) $\varphi_k \vee \bigvee_{l \leq i \leq k} \overline{\delta}_i$ |
| $\mathsf{F}\,\sigma$ | $\varphi_i \rightarrow \sigma_i \vee \varphi_{i+1}$ | ✓* | (n) $\overline{\varphi}_i \vee \sigma_i \vee \varphi_{i+1}$ |
| | $\sigma_i \rightarrow \varphi_i$ | ✓ | (o) $\overline{\sigma}_i \vee \varphi_i$ |
| | $\varphi_{i+1} \rightarrow \varphi_i$ | ✓ | (p) $\overline{\varphi}_{i+1} \vee \varphi_i$ |
| | $\varphi_l \rightarrow \bigvee_{l \leq i \leq k} \sigma_i$ | | (q) $\overline{\varphi}_l \vee \bigvee_{l \leq i \leq k} \sigma_i$ |
| $\mathsf{G}\,\sigma$ | $\varphi_i \rightarrow \sigma_i$ | ✓ | (r) $\overline{\varphi}_i \vee \sigma_i$ |
| | $\varphi_i \rightarrow \varphi_{i+1}$ | ✓ | (s) $\overline{\varphi}_i \vee \varphi_{i+1}$ |
| | $\neg\varphi_i \rightarrow \neg\sigma_i \vee \neg\varphi_{i+1}$ | ✓ | (t) $\varphi_i \vee \overline{\sigma}_i \vee \overline{\varphi}_{i+1}$ |
| | $\neg\varphi_k \rightarrow \bigvee_{l \leq i \leq k} \neg\sigma_i$ | | (u) $\varphi_k \vee \bigvee_{l \leq i \leq k} \overline{\sigma}_i$ |
| $\delta\,\mathsf{R}\,\sigma$ | $\varphi_i \rightarrow \sigma_i$ | ✓ | (v) $\overline{\varphi}_i \vee \sigma_i$ |
| | $\varphi_i \wedge \neg\varphi_{i+1} \rightarrow \delta_i$ | ✓ | (w) $\overline{\varphi}_i \vee \varphi_{i+1} \vee \delta_i$ |
| | $\sigma_i \wedge \delta_i \rightarrow \varphi_i$ | ✓ | (x) $\overline{\sigma}_i \vee \overline{\delta}_i \vee \varphi_i$ |
| | $\varphi_{i+1} \wedge \sigma_i \rightarrow \varphi_i$ | ✓ | (y) $\overline{\varphi}_{i+1} \vee \overline{\sigma}_i \vee \varphi_i$ |
| | $\bigwedge_{l \leq i \leq k} \sigma_i \rightarrow \varphi_l$ | | (z) $\varphi_l \vee \bigvee_{l \leq i \leq k} \overline{\sigma}_i$ |

from Theorem 1 and the fact that we weaken the restrictions regarding the signal's values in the process.

Please note that while for Theorem 1 we considered operators $\mathsf{F}$, $\mathsf{G}$, $\mathsf{W}$, and $\mathsf{R}$ as syntactic sugar (that can be rewritten easily), this is not favorable for both performance reasons and the model-based diagnosis approach presented in Section 5. Instead, we directly translate these operators, reducing the amount of variables and clauses. This also allows us to provide diagnoses addressing the original operators (no need for rewriting). Table 2 lists the corresponding rationale and clauses. For $\varphi = \delta\,\mathsf{W}\,\sigma$, the argumentation regarding correctness is roughly the same as for $\delta\,\mathsf{U}\,\sigma$, with two exceptions: According to Def. 3, there is no need for $\sigma$ to hold eventually. Thus, regarding the second direction, we can remove the original Clause (i) that limits the postponement. We have to add a new Clause (m) for the first direction though, that catches the situation of infinite postponement ($\mathsf{G}\,\delta$), and enables $\varphi_k$ accordingly. The operator $\mathsf{F}\,\sigma$ is a specific case of the until operator $\mathsf{U}$ with $\delta = \top$, where, compared to $\mathsf{U}$, we don't need to instantiate Clause (n) (merging ($f_1$) and ($f_2$)) in the loop. In this specific case $\varphi_i$ does not change in the loop, and is propagated by Clause (p) and the fact that $\tau_{k+1} = \tau_l$ (so that we also use $\overline{\varphi}_l$ instead of $\overline{\varphi}_k$ in Clause (q)). The release operator is similar to the weak until, but $\delta$ would have to overlap with $\sigma$ for one step, where due to lack of space we have to leave the details to the reader.

Prior to tackling diagnosis in Section 5, we evaluate this basic encoding in Section 4.

## 4 EVALUATION

We implemented our approach in Python (CPython 2.7.1) and compared it against using the state-of-the-art NuSMV model checker (Cimatti *et al.*, 2002) in its latest official version 2.5.4, both in BDD and SAT (Bounded Model Checking – BMC) mode, denoted in the results as "NuSMV-BDD" and "NuSMV-BMC" respectively. Following the suggested encoding for a path in (Heljanko *et al.*, 2005), we encoded the trace for NuSMV as follows, and passed along $k$ and $l$ for the BMC variant for a fair comparison:

$$\forall 0 \leq i \leq k : (\tau_i \texttt{ \& } s_i \texttt{ \& next(}\tau_{i+1} \texttt{ \& } s_{i+1}\texttt{))}$$

Our version of NuSMV-BMC uses MiniSAT (Eén and Sörensson, 2003) version 2.0(-070721) as SAT solver, which unfortunately is unable to compute minimal unsatisfiable cores (MUC) as needed by our model-based diagnosis approach in Section 5. We therefore evaluated our encoding using three different SAT solvers, namely the MiniSAT 2.0 version used by NuSMV, its latest version 2.2 and the most recent MUC-capable PicoSAT (Biere, 2008) version 936. While the MiniSAT versions were compiled to a CPython extension using the Boost.Python library, we used the standard file-based interface for PicoSAT.

Two test sets containing 27 common and 1000 random formulae respectively were taken from (Somenzi and Bloem, 2000), complemented by our own samples for scalability tests. The traces were generated such that every $p \in AP$ is true at each time step with a probability of 0.5, and a trace's stem and loop are equally sized (i.e., $l = k/2$). We executed our tests on an early 2011-generation MacBook Pro (Intel Core i5 2.3GHz, 4GiB RAM, SSD) running an up-to-date version of Mac OS X 10.6 with the GUI and swapping disabled. The reported run-times represent total user-experienced time, i.e. include the encoding time and the backend's (NuSMV/PicoSAT/MiniSAT) execution time. For the BMC-based NuSMV variant, the number of instantiated variables and clauses was recorded by dumping it's internal DIMACS file afterwards.

Figure 1 shows the run-times for the set of 27 common formulae taken from (Somenzi and Bloem, 2000) and a set of 100 random traces with $k = 100$, where the formulae's size ranges from 3 to 22. Considering the average of the reported run-times, our encoding using the fastest solver (LTL-SAT/MiniSAT-2.2) outperforms NuSMV(-BDD) by a factor of about 12. Using PicoSAT, run-times are about 1.6 times higher than those obtained with MiniSAT-2.2, where this can presumably be ascribed to the file-based interface. Nevertheless, also in this case, we outperform the fastest NuSMV(-BDD) variant by a factor of about 7, which itself outperforms the NuSMV-BMC variant. While the latter is counterintuitive at first, it seems that for our examples the NuSMV-BMC SAT encoding produces too much overhead even for fixed values of $k$ and $l$ (cf. Table 3). For our encoding, we observe that MiniSAT version 2.2 has just a very slight performance advantage over the one used by NuSMV.

As solving the SAT problem $E(\varphi, \tau)$ provides the subformulae's evaluation only in the case of a witness (a satisfiable instance), we report also the run-times for $E(\neg\varphi, \tau)$, suffixed ($\neg$) in the graphs, The variant with the suffix +U automatically derives for unsatisfiable $E(\varphi, \tau)$ a full counterexample trace by reusing the encoding when extending it to $E(\neg\varphi, \tau)$, and offers run-times only slightly above the run-times for $E(\varphi, \tau)$ only. This is due to the fact that the construction of the clauses takes quite some percentage of the total time. Figure 2 further highlights that, offering the

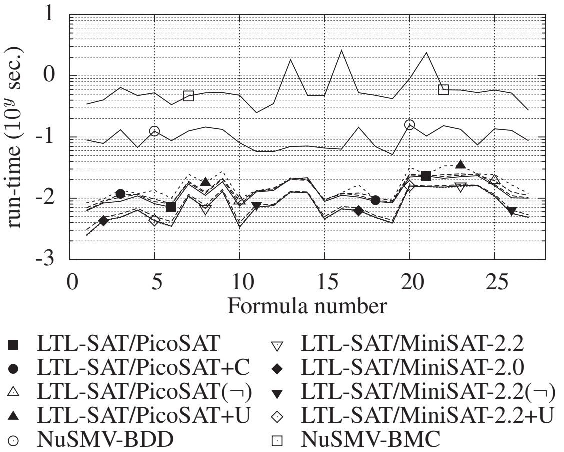


Figure 1: Run-times for a test set containing 27 common formulae taken from (Somenzi and Bloem, 2000).

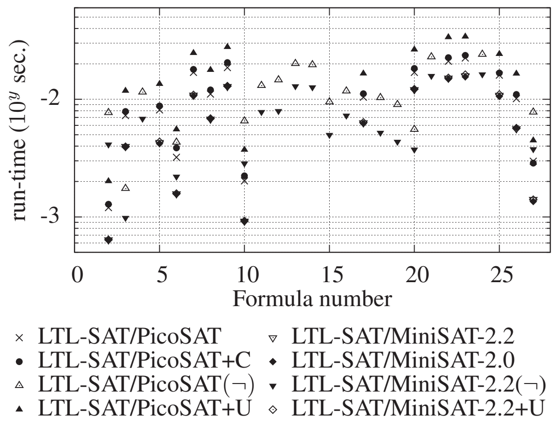


Figure 2: Run-times for unsatisfiable specification/trace combinations from Figure 1.

run-times for the subset of samples with unsatisfiable combinations of $\varphi$ and $\tau$. Deriving a minimal unsatisfiable core (the PicoSAT variant with the suffix +C) as used for diagnosis (see next section) requires very little overhead, so that the curve is nearly indistinguishable from the standard one (no suffix at all).

Figure 3 confirms the performance advantage of our encoding against NuSMV-BDD (due to the previous results, we excluded NuSMV-BMC), showing the run-times for a test set of 1000 random formulae from (Somenzi and Bloem, 2000) and 100 random traces ($k = 100$). While the variants using MiniSAT are faster than the PicoSAT ones by a factor of about 1.8, the average run-time benefit of LTL-SAT/PicoSAT against NuSMV is with a factor of 5 a bit lower though.

For a comparison of the SAT encodings, Table 3 lists the average number of produced variables and clauses for 10 random formulae per line (created using the method in (Daniele *et al.*, 1999) – $N = \lfloor |\varphi|/3 \rfloor$ variables, uniform distribution of LTL operators), each verified against 10 random traces. While NuSMV-BMC sometimes uses less variables for small traces (see, e.g., $|\varphi| = 50$, $|\tau| = 10$), both the number of variables and the number of clauses scale disproportionately high when extending the trace.

Figure 4 shows the run-times when scaling the specification size and using the same random formulae concept as for Table 3. The graph suggests a sub-exponential scaling of our encoding in contrast to a super-exponential scaling of NuSMV-BDD, where the trend for the latter could not be verified due to memory problems for $|\varphi| > 60$. In fact, NuSMV used more than 2 GiB (1 GiB = $2^{30}$ bytes versus 1 GB = $10^9$ bytes) of memory for some samples with $|\varphi|$ just over 60, while with PicoSAT we solved them using only 40 MiB. Even for the last sample in the test set ($|\varphi| = 300$) we got along using 350 MiB, a value topped by NuSMV already at $|\varphi| = 50$.

In Figure 5, we report some results regarding trace scalability. For a random specification of size 20

$$((\mathsf{X}\, v_1)\,\mathsf{R}\,((v_2\,\mathsf{U}\,\mathsf{X}\,((\mathsf{X}\, v_4)\,\mathsf{U}\,(v_2\,\mathsf{U}\, v_2)))\,\mathsf{U}\, v_4))\,\mathsf{U}\,!(v_4\,\mathsf{R}\,!v_4),$$

we report an average run-time over 10 random traces (with stem and loop equally sized) for any trace length

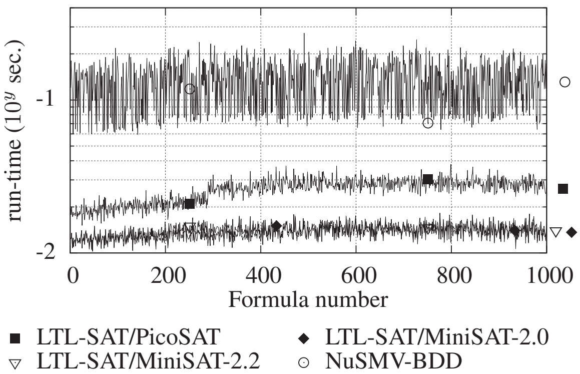


Figure 3: Run-times for a test set containing 1000 common formulae from (Somenzi and Bloem, 2000).

in the range [10,1000]. Both approaches scaled similarly, with LTL-SAT/PicoSAT being about 4 times faster for trace lengths above approx. 200. Regarding memory, NuSMV-BDD used about 193 MiB for a trace of length 1000, while our approach (including PicoSAT) got along with approx. 50 MiB.

## 5 MODEL-BASED DIAGNOSIS OF LTL SPECIFICATIONS FOR SPECIFIC SCENARIOS

While we saw that with the basic encoding we can easily verify a trace's containment in a specification and derive/complete witnesses and counterexamples with given bounds $k$ and $l$, there is always a motivation for retrieving an explanation if things are different than expected. To this purpose, we propose the following instrumentation of our encoding, enabling a model-based diagnosis (Reiter, 1987; de Kleer and Williams, 1987) of LTL specification flaws.

Like Reiter (see Section 2), we propose to add special predicates that encode specific behavioral assumptions. In case of an LTL formula, we propose to add such a predicate $op$ for any operator in the formula, with the purpose to toggle the assumption whether the correct operator was used or not. While assumptions on a signal's value should be instantiated for each time

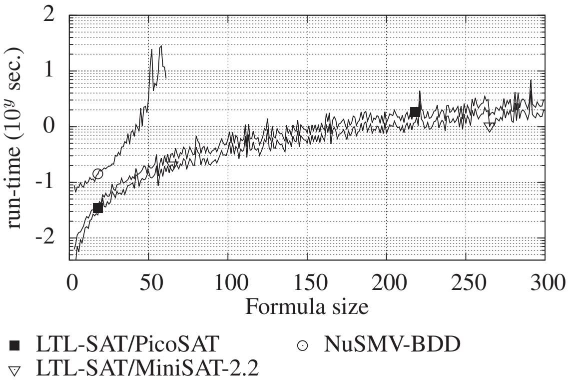


Figure 4: Scaling $\varphi$ ($\tau : k = 100, l = 50$): Average run-times for 20 formulae and 10 traces.

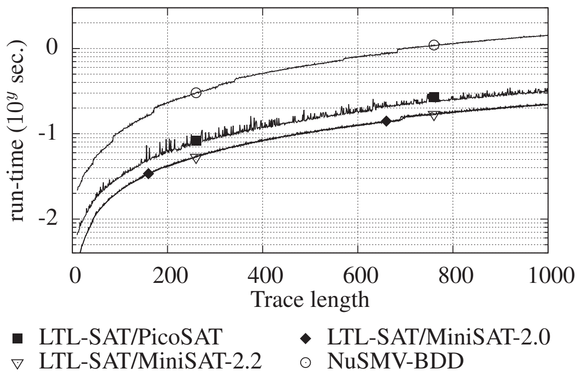


Figure 5: Scaling $\tau$ ($|\varphi| = 20, l = k/2$): Average run-times for 10 traces.

Table 3: Number of variables and clauses for different input sizes (each line averages over 100 traces for 10 formulae).

| $\lvert\varphi\rvert$ | $\lvert\tau\rvert$ | LTL-SAT/PicoSAT | | | NuSMV-BMC | | |
|---|---|---|---|---|---|---|---|
| | | run-t. | #V | #C | run-t. | #V | #C |
| 10 | 10 | 0.0061 | 141.9 | 290.4 | 0.0237 | 294.98 | 707.74 |
| | 20 | 0.0068 | 243.6 | 503.7 | 0.0374 | 1015.65 | 2842.90 |
| | 50 | 0.0103 | 596.7 | 1347.4 | 0.1444 | 5567.61 | 16920.49 |
| | 100 | 0.0157 | 1181.7 | 2614.5 | 0.5795 | 25214.62 | 79376.47 |
| 20 | 10 | 0.0077 | 276.1 | 584.7 | 0.0255 | 304.86 | 830.95 |
| | 20 | 0.0108 | 529.2 | 1173.7 | 0.0433 | 1019.64 | 3302.27 |
| | 50 | 0.0172 | 1178.1 | 2676.2 | 0.2928 | 6524.30 | 25360.56 |
| | 100 | 0.0281 | 2444.2 | 5548.5 | 0.7812 | 26221.50 | 100382.14 |
| 50 | 10 | 0.0135 | 657.8 | 1412.7 | 0.0337 | 464.03 | 1576.67 |
| | 20 | 0.0201 | 1323.0 | 2968.3 | 0.0779 | 1500.28 | 6786.80 |
| | 50 | 0.0389 | 3253.8 | 7360.6 | 0.3166 | 6640.64 | 37741.57 |
| | 100 | 0.0697 | 6150.9 | 14295.7 | 1.1591 | 20152.24 | 121500.65 |
| 100 | 10 | 0.0232 | 1327.7 | 2879.0 | 0.0477 | 735.02 | 2847.97 |
| | 20 | 0.0370 | 2553.6 | 5773.0 | 0.1180 | 1943.21 | 11540.14 |
| | 50 | 0.0710 | 6109.8 | 14065.8 | 0.5146 | 8396.00 | 65230.12 |
| | 100 | 0.1436 | 12594.7 | 29143.9 | 2.6496 | 30223.90 | 289962.09 |

step, we propose a single assumption for all time steps for the operators (a specification $\varphi$ usually does not vary over time, while signals do). Furthermore, while 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, this could be counterproductive in a diagnostic context. This is due to situations when only one instance is at fault (e.g. should rather be $a \,\mathsf{W}\, b$ instead of $a \,\mathsf{U}\, b$). To that purpose, we instrument any clause resulting from some operator $\varphi$ with the negated corresponding predicate $\overline{op_\varphi}$ as further disjunct, and add another clause $op_\varphi$ stating that the predicate should be $\top$. Clause (c$_1$) $\varphi_i \vee \overline{\delta}_i$ for a logic OR, for example, would be instrumented to $\overline{op_\varphi} \vee \varphi_i \vee \overline{\delta}_i$.

Using the trace as observation $OBS$, i.e. taking it as granted, we can then implement a model-based diagnosis approach on the specification $SD$ as follows: Using a SAT solver that is able to return minimal unsatisfiable cores, we extract those clauses from the returned core that assign the operator assumptions. These represent the conflict sets interfacing our problem domain with classic model-based diagnosis algorithms (Reiter, 1987; de Kleer and Williams, 1987).

For our running example of the two-line arbiter, we can provide the essential diagnostic data to identify the source of the unexpected result. In less than 0.1 seconds on our test platform (see Section 4), we obtained the following diagnoses using HS-DAG, where the node numbers $n_i$ refer to the subformulae as depicted in Figure 6.

$$\{\{n_7\}, \{n_8\}, \{n_{12}\}, \{n_{19}\}, \{n_{27}\}\}$$

There are only single fault diagnoses for this scenario, and obviously, if some subformula $\delta$ is a diagnosis, also the enclosing formulae are diagnoses. Note that the nodes $n_0$ and $n_1$ cannot be diagnoses as they encode the mandatory *and* operators combining the various requirements in the specification. Intuitively, it thus is a good idea to prioritize in such chains those subformulae farthest from the root. Therefore, the until operator in $n_{19}$ and the corresponding subformula $\neg g_1 \,\mathsf{U}\, r_1$ from $R_4$ (at this location, the designer should have used a weak until $\mathsf{W}$ instead of a strong until $\mathsf{U}$) should be the first candidates for inspection.

Similarly to instrumenting the specification, we could take the specification as granted (with no instrumenting assumptions) and instrument the trace in order to ask what is wrong with the trace for some encountered inconsistency. An obvious instrumentation in that respect is that of adding a predicate for each signal's value at any step. More elaborate fault models in that respect should help reduce the number of predicates, which is important as the diagnosis space is exponential in the number of (assumption) predicates.

Reiter's MBD algorithm and our encoding of above consider a very general (weak) fault model, which can be refined by implementing fault modes as suggested in (de Kleer and Williams, 1989). In that case each assumption defines a behavioral (sub-)mode $\in \{nominal, fault_1, \ldots, fault_{n-1}\}$ where for any mode the actual behavior has to be defined. Such an assumption would be encoded with multiple predicates (ld($n$)) that, due to the structure-preserving encoding, can easily toggle between the corresponding local clauses. Adopting a non-structure preserving idea like (Biere *et al.*, 1999) for our setting and instrumenting it accordingly, would obviously result in a tremendous blowup.

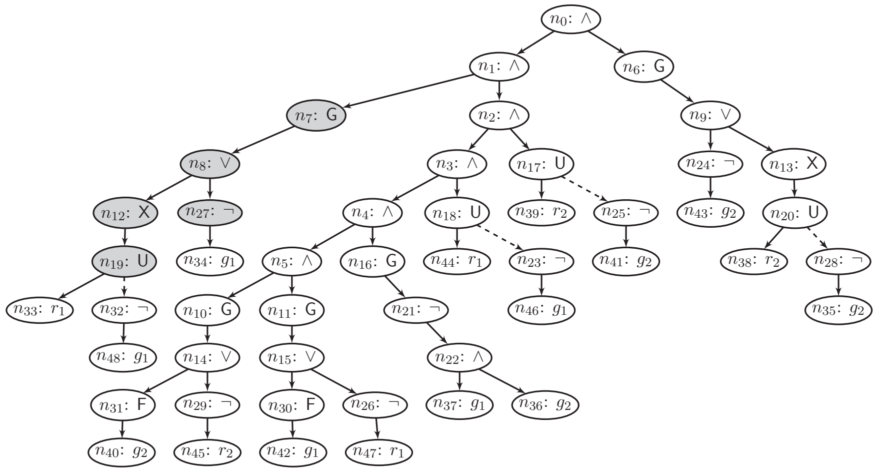


Figure 6: Syntax tree of the arbiter specification (shaded nodes represent diagnoses, dashed edges point to the left subformula where relevant). Total diagnosis time: $< 0.1$s (HS-DAG, encoding, theorem prover (PicoSAT), ...).

A good example for an effective fault mode for the strong until operator (U) is suggested by our running example; replace the strong with a weak until (W). Due to the similarity of the clauses, this would require only to toggle between Clauses (i) and (m).

For a proof-of-concept, we implemented a couple of fault modes addressing functional errors and "typos":

- confusion of the Boolean operators $\wedge$ and $\vee$,
- confusion of the unary operators F, G and X,
- confusion of the binary operators U, W and R,
- twisted $\delta$ and $\sigma$ for U, W and R,
- wrong variable used.

Adopting a notion of conflicts similar to the one presented in (Nyberg, 2011), we made HS-DAG aware of operator fault modes for the search space exploration. On our test platform (see Section 4), we obtained for our example in less than 0.4 seconds the 9 diagnoses listed in Table 4, where the correct diagnosis ($\Delta_7$: "replace $\neg g_1 \,\mathsf{U}\, r_1$ by $\neg g_1 \,\mathsf{W}\, r_1$") is among them.

While we just started optimizing for large samples, first scalability tests show that even for problem size $|\varphi| = |\tau| = 200$ *all* fault mode-based single-fault diagnoses can be obtained within reasonable limits ($239/266.3/302$ diagnoses in $2395/2462/2521$ seconds, using $825.8/831.8/834.4$ MiB – min/avg/max over 10 samples with approximately $5 \cdot 10^4$ variables and $1.6 \cdot 10^6$ clauses per sample).

## 6 CONCLUSIONS

We proposed a structure-preserving SAT encoding for LTL in the context of behavioral samples, and its instrumentation aiming at a model-based diagnosis of LTL-specification flaws. While a structure-preserving SAT encoding implementing an expansion rule-based temporal unrolling is not new (Heljanko *et al.*, 2005; Schuppan, 2012), we show how to efficiently catch obligations pushed in time in the context of specific traces with given length and loop-back time step. We furthermore show an easy instrumentation regarding a model-based diagnosis approach using a general fault model, so that a designer can be provided

Table 4: The nine fault mode-based diagnoses for the arbiter example. They concern requirement $R_4$ only. Total diag. time: $< 0.4$s (HS-DAG, encoding, TP, ...).

| | |
|---|---|
| Original | $\mathsf{G}\,(g_1 \rightarrow \mathsf{X}\,(\neg g_1 \,\mathsf{U}\, r_1))$ |
| $\Delta_1$ | $\mathsf{X}\,(g_1 \rightarrow \mathsf{X}\,(\neg g_1 \,\mathsf{U}\, r_1))$ |
| $\Delta_2$ | $\mathsf{G}\,(g_1 \rightarrow \mathsf{F}\,(\neg g_1 \,\mathsf{U}\, r_1))$ |
| $\Delta_3$ | $\mathsf{G}\,(g_1 \rightarrow \mathsf{X}\,(r_1 \,\mathsf{R}\, \neg g_1))$ |
| $\Delta_4$ | $\mathsf{G}\,(g_1 \rightarrow \mathsf{X}\,(\neg g_1 \,\mathsf{U}\, r_2))$ |
| $\Delta_5$ | $\mathsf{F}\,(g_1 \rightarrow \mathsf{X}\,(\neg g_1 \,\mathsf{U}\, r_1))$ |
| $\Delta_6$ | $\mathsf{G}\,(g_1 \rightarrow \mathsf{X}\,(\neg g_1 \,\mathsf{U}\, g_2))$ |
| $\Delta_7$ | $\mathsf{G}\,(g_1 \rightarrow \mathsf{X}\,(\neg g_1 \,\mathsf{W}\, r_1))$ |
| $\Delta_8$ | $\mathsf{G}\,(g_1 \rightarrow \mathsf{X}\,(r_1 \,\mathsf{U}\, \neg g_1))$ |
| $\Delta_9$ | $\mathsf{G}\,(g_1 \rightarrow \mathsf{X}\,(r_1 \,\mathsf{W}\, \neg g_1))$ |

with diagnoses regarding her specification. In contrast to (Schuppan, 2012), a designer can define concrete scenarios and ask concrete questions (via the trace, like the interface to Property Simulation in (Pill *et al.*, 2006; Bloem *et al.*, 2007a)) and is presented with (multi-fault) diagnoses in terms of the operators in the specification, rather than unsatisfiable cores of clauses or transitions in an automaton. While we used Reiter's classic HS-DAG for our tests, our encoding obviously supports also newer approaches like (Stern *et al.*, 2012), and can be used in approaches computing diagnoses directly (Metodi *et al.*, 2012). We showed how fault modes can be integrated and provided some suggestions regarding relevant fault models. Our evaluation and the examples show the attractiveness of our approach. An alternative research direction would be to use derived automata (or their symbolic implementation (Bloem *et al.*, 2007b)) as model and adopt research in the context of discrete event systems (Grastien *et al.*, 2012). Incorrect/alternative operators for diagnoses at operator level would however be cumbersome to derive, for example, from sets of faulty transitions of a (for computation reasons) heavily optimized automaton with hundreds of states and transitions. This was one of the main motivations for our current research. Future work will have to identify effective fault mode sets (i.e. requiring an extensive evaluation) in order to offer detailed diagnoses within reasonable resource limits. Regarding runtime verification, also the consideration of finite stems in a diagnostic context seems interesting. Future research will also cover constructs from more elaborate logics like the Property Specification Language (PSL) (Eisner and Fisman, 2006).

## ACKNOWLEDGEMENTS

Our research presented in this paper resulted from the project *MoDiaForTeD* funded by the Austrian Science Fund (FWF): P22959-N23. See http://modiaforted.ist.tugraz.at for more information.

## REFERENCES

(Biere *et al.*, 1999) A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *5th 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.*, 2007a) 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.

(Bloem *et al.*, 2007b) R. Bloem, A. Cimatti, I.Pill, and M. Roveri. Symbolic implementation of alternating automata. *Int. Journal of Foundations of Computer Science (IJFCS)*, 18(04):727–743, 2007.

(Cimatti *et al.*, 2002) A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV version 2: An opensource tool for symbolic model checking. In *Computer-Aided Verification*, pages 241–268, 2002.

(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 *11th International Joint Conference on Artificial Intelligence*, pages 1324–1330, 1989.

(Eén and Sörensson, 2003) N. Eén and N. Sörensson. Minisat v1.13–a SAT solver with conflict-clause minimization. In *6th International Conference on Theory and Applications of Satisfiability Testing*, pages 502–518, 2003.

(Eisner and Fisman, 2006) C. Eisner and D. Fisman. *A Practical Introduction to PSL.* Springer, 2006. ISBN: 978-0-387-35313-5.

(Fisman *et al.*, 2009) D. Fisman, O. Kupferman, S. Sheinvald-Faragy, and M.Y. Vardi. A framework for inherent vacuity. In *4th International Haifa Verification Conference on Hardware and Software: Verification and Testing*, pages 7–22, 2009.

(Grastien *et al.*, 2012) A. Grastien, P. Haslum, and S. Thiébaux. Conflict-based diagnosis of discrete event systems: theory and practice. In *13th Int. Conf. on the Principles of Knowledge Representation and Reasoning*, pages 489–499, 2012.

(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 *17th International Conference on Concurrency Theory*, pages 37–51, 2006.

(Liffiton and Sakallah, 2005) M. Liffiton and K. Sakallah. On finding all minimally unsatisfiable subformulas. In *Theory and Applications of Satisfiability Testing*, pages 34–42, 2005.

(Metodi *et al.*, 2012) A. Metodi, R. Stern, M. Kalech, and M. Codish. Compiling model-based diagnosis to boolean satisfaction. In *26th AAAI Conference on Artificial Intelligence*, 2012. (to appear).

(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 *et al.*, 2006) I. Pill, S. Semprini, R. Cavada, M. Roveri, R. Bloem, and A. Cimatti. Formal analysis of hardware requirements. In *43rd Conference on Design Automation*, pages 821–826, 2006.

(Pnueli, 1977) A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57, 1977.

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

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

(Somenzi and Bloem, 2000) F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *12th 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 *26th AAAI Conference on Artificial Intelligence*, 2012. (to appear).

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