Exploiting Parse Trees in LTL Specification Diagnosis

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

Abstract

Specifications are a development process' lifeblood. Capturing the designers' intentions regarding functionality, interface, test targets, and other aspects, they establish the correct context in design communication, development, and verification amongst other steps like synthesis. A specification's quality is thus a crucial factor. Recently we showed a way to exploit model-based diagnosis for the development of formal (functional) specifications in the Linear Temporal Logic (LTL). In this paper we show how to improve that diagnosis process' search via considering a specification's parse tree. Implementing our ideas with the well-established HS-DAG algorithm, we report experimental results showing our reasoning's attractiveness.

1 Introduction

Specifications capturing the design intent are an essential instrument in any design process. As the basic vehicle for establishing the context in design-related communication, specifications may define a system's functionality, interfaces, test goals, and many other aspects. Consequently, they drive a system's creation, testing, verification, and potentially even a synthesis process. While it is thus not unexpected that up to 50 percent of product defects, and up to 80 percent of rework efforts can be traced back to requirement defects [Wiegers, 2001], these figures illustrate the high demand for tools and means that help us in getting our specifications right.

Drawing on the precise syntax and unambiguous semantics of formal functional specifications and their languages, coverage and vacuity concepts emerged [Fisman *et al.*, 2009; Kupferman, 2006] that can help in identifying specification flaws. For an enhanced user experience, formal specification development tools like RAT [Pill *et al.*, 2006] offer workflows that allow a designer to explore and verify a specification's semantics. Complementing such work, we showed recently a way to exploit model-based diagnosis for the development of specifications in the Linear Temporal Logic (LTL) [Pill and Quaritsch, 2013]. Based on behavioral samples (traces) that *unexpectedly* satisfy (witnesses) or contradict (counterexamples) a specification, we can isolate corresponding diagnoses at a specification's operator level regarding the root causes for the encountered issue. As underlying reasoning engine for the verification of diagnostic theories, we use a SAT solver and a corresponding encoding of the specification and the trace.

In the context of weak fault models, we show in this paper how to exploit structural information about a specification (i.e. its parse tree) for speeding up the computation of these diagnoses. That is, inspired by the concept of dominance defined for flow-graphs [Prosser, 1959; Lengauer and Tarjan, 1979] that has been exploited also for digital circuits [Kirkland and Mercer, 1987], we show how to draw on the intuitive observation that "If some subformula can resolve a conflict, this is true also for its parent".

While in the context of our specification models as established in [Pill and Quaritsch, 2013], this offers us no option for design abstraction resulting in smaller models, nor to statically restrict the diagnosis space, we exploit this observation dynamically in the diagnosis algorithm, i.e. the structured search itself. We implemented our reasoning with the well-known HS-DAG algorithm, that is [Greiner *et al.*, 1989]'s version of Reiter's diagnosis algorithm [Reiter, 1987], and report first results regarding the effects in this paper.

Our paper is structured as follows. In Section 2, we cover the preliminaries for our approach as discussed in Section 3. After showing in Section 4 how to implement our reasoning for HS-DAG, we report experimental results in Section 4.1. Section 5 offers a discussion of our approach and future work, as well as corresponding conclusions. Related work is discussed throughout the paper where appropriate.

2 Preliminaries

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 $\Sigma = 2^{AP}$.

Definition 1. *Let AP be a finite set of atomic propositions, and δ and φ LTL formulae. Then an LTL formula 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*

Definition 2. *A* parse tree *(*syntax tree*) $\mathcal{T}(\varphi) = (V_\varphi, v_\varphi, E_\varphi, l(v \in V_\varphi))$ for an LTL formula φ is a directed, vertex-labeled tree, where*

- *V_φ is the set of vertices such that for each subformula ψ in φ there is exactly one vertex (v_ψ) labeled with ψ,*
- *$l(v)$ is a labeling function for vertices $v \in V_\varphi$ s.t. $l(v_\psi) = \psi$,*

- $v_\varphi \in V$ *is the single root vertex,*
- *and E is $\mathcal{T}$'s set of edges, s.t. for $v_{\psi_1}, v_{\psi_2} \in V_\varphi$, $e = (v_{\psi_1}, v_{\psi_2})$ is in E, iff ψ_2 is an operand of ψ_1.*

Note that we consider multiple occurrences of a syntactic construct to be multiple subformulae. With $\top$ denoting logic *True/High/1* and $\bot$ denoting logic *False/Low/0*, the popular operators $\delta\,\mathsf{R}\,\sigma$, $\mathsf{F}\,\varphi$, $\mathsf{G}\,\varphi$, and $\delta\,\mathsf{W}\,\sigma$ 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. While we showed in [Pill and Quaritsch, 2013] how to encode these operators directly for our LTL SAT encoding (see, e.g., Def. 7), without loss of generality, we focus on the core of LTL in this paper.

LTL is defined in the context of infinite traces, which we define as explicit finite sequences as is usual. Such a finite sequence of length k can describe a single infinite trace only in the form of a lasso-shaped trace (with a cycle looping back from the last step k to $0 \leq l \leq k$). The other option would be to consider the sequence as a prefix, such that it refers to the set of infinite traces that extend it. Note that we always refer to an infinite trace when using the term trace.

Definition 3. *An infinite trace τ is an infinite word over letters from some alphabet Σ 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 $0 \leq i \leq k$ and $(...)^\omega$ denoting infinite repetition of the corresponding (sub-)sequence. With τ^i, we refer to τ's suffix starting with τ_i.*

The LTL core operators' semantics in the context of infinite traces are defined as follows:

Definition 4. *Given a trace τ and an LTL formula φ, $\tau(=\tau^0)$ satisfies φ, denoted as $\tau \models \varphi$, under the following conditions*

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

Regarding diagnostic reasoning, we adopt for our presentation the formalizations of Reiter's consistency-oriented theory of diagnosis [Reiter, 1987]. Given a system's set of components *COMP*, assumption predicates $AB(c_i)$ for all $c_i \in$ *COMP* encoding whether c_i behaves *abnormally*, a system description *SD* defining the correct behavior $\neg AB(c_i) \Rightarrow \mathit{NominalBehavior}(c_i)$, and some actual observations *OBS* about a system's behavior, the system is considered to be at fault iff $SD \cup OBS \cup \{\neg AB(c_i) | c_i \in COMP\}$ is inconsistent. While a minterm in the assumptions defines a specific state of the system, a diagnosis Δ is a subset-minimal set of faulty components that explains the inconsistency and is a subset of at least one "consistent" minterm.

Definition 5. *A* 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 set of diagnoses as the minimal hitting sets of the set *CS* of (not necessarily minimal) conflicts for (*SD*, *COMP*, *OBS*).

Definition 6. *A* conflict *C for (SD, COMP, OBS) is a set $C \subseteq COMP$ such that $SD \cup OBS \cup \{\neg AB(c_i) | c_i \in C\}$ is inconsistent. Iff there is no $C' \subset C$, such that C' is a conflict, C is a minimal conflict.*

If not available a priori, Reiter's algorithm is able to derive *CS* on-the-fly, which is an attractive feature in situations where the cardinality of diagnoses is restricted so that only a subset of *CS* has to be computed. For the organization of a structured search, the algorithm creates a node- and edge-labeled tree by iteratively expanding its nodes in breadth-first order, starting with the root node n_0. With $h(n) \subseteq COMP$ the set of edge labels on a node n's path from the root ($h(n_0) = \emptyset$), each non-leaf node n is labeled with a conflict C_i s.t. $C_i \cap h(n) = \emptyset$. If not computed a priori, valid C_is are requested from the theorem prover for negative checks whether $h(n)$ could be a diagnosis (i.e. whether $SD \cup OBS \cup \{\neg AB(c_i) | c_i \in COMP \setminus h(n)\}$ is consistent). Affirmative checks (or the absence of such a C in a pre-computed *CS*) define n as leaf (labeled ✓) and then $h(n)$ is considered to be a potential diagnosis. Specific rules pruning the tree ensure that the "final" leafs are indeed (subset-minimal) diagnoses. New nodes are created in a breadth-first manner by constructing for each c_j in a non-leaf node's label C_i an outgoing edge labeled c_j and a corresponding destination node.

When the search for diagnoses is limited to some bound k regarding their cardinality, nodes with $|h(n)| = k$ need not be expanded. [Greiner *et al.*, 1989] corrected Reiter's original formulations that had some minor but serious flaws, and offer an improved algorithm, HS-DAG, that uses a directed acyclic graph (DAG) instead of a tree.

In [Pill and Quaritsch, 2013] we showed a way to exploit model-based diagnosis using weak or strong fault models for the diagnosis of LTL specifications given behavioral samples (traces). In order for the paper to be self-contained, we rehearse the necessary definitions and one theorem from [Pill and Quaritsch, 2013].

Definition 7. *[Pill and Quaritsch, 2013] 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. Note that we assume k and l to be 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 8. *[Pill and Quaritsch, 2013] For a given infinite trace τ (with given k), $E_2(\tau) = \bigwedge_{0 \leq i \leq k} \Big[\bigwedge_{p_i \in \tau_i} p_i \wedge \bigwedge_{p_i \in AP \setminus \tau_i} \neg p_i\Big]$ encodes the signal values as specified by τ.*

Theorem 1. *[Pill and Quaritsch, 2013] 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 7 and 8 is satisfiable, $SAT(E_{WFM}(\varphi, \tau))$, iff $\tau \models \varphi$ under assumptions op.*

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

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 ($0 \leq i \leq k$).

Using Theorem 1, we can determine for some unexpected counterexample or witness via $E_{WFM}(\varphi, \tau)$ or $E_{WFM}(\neg\varphi, \tau)$ the corresponding diagnoses in the specification's operators/subformulae [Pill and Quaritsch, 2013].

3 Exploiting a Specification's Parse Tree during Behavioral LTL Diagnosis

In this section we show how to exploit an LTL specification φ's actual parse tree in the search for diagnoses as of Theorem 1. Without loss of generality, we occasionally refer for our argumentation to the minimal conflicts that can characterize a diagnosis problem as of Reiter's diagnosis theory. The possible effects are discussed in the context of the well-known HS-DAG algorithm due to the easily graspable DAG that encodes its search. Our reasoning is however general enough, so that the underlying ideas transfer also to other diagnosis algorithms. The main observation our reasoning draws on is covered in Proposition 1.

Proposition 1. *If some subformula ψ from specification φ can resolve an issue (i.e. a minimal conflict), then so can all the superformulae of ψ.*

The correctness of this intuitive observation is easily shown considering our encoding for Theorem 1. When a subformula ψ is considered abnormal, the corresponding time-instantiated variables ψ_i are freed in that their values become undefined. Via the individual subformulae's encodings, the *chosen* values for ψ_i however still influence (depending on the operators) the evaluation of the variables of its superformulae (those formulae on the path from root vertex v_φ to v_ψ). Thus, if there is some satisfying assignment for ψ_i when considering ψ faulty, this assignment is still a satisfying one when considering a superformula δ to be at fault, and freeing the assignments of $\delta' s$ subformulae (not signals!) that are irrelevant for the evaluation of φ anyway (due to $AB(\delta)$).

Our observation in Proposition 1 is also reflected in the minimal conflicts describing the diagnosis problem:

Proposition 2. *If a minimal conflict C_i contains some subformula ψ, then it contains also all its superformulae.*

The correctness of this proposition is easy to see. Superformula δ's not being in C would contradict Proposition 1 that δ can resolve/hit (at least) those minimal conflicts that ψ can resolve (including C_i). Note that while the proposition is obviously not true for a non-minimal C_i, even such a conflict might be pruned regarding subformulae where not all of its superformulae are in C_i (shedding some of the non-minimal/unnecessary components).

In the following we show how to use these propositions in order to derive new facts from already computed ones. That is, for instance, from some diagnosis Δ we can derive further diagnoses Δ'.

Lemma 1 (Infer-up). *For a diagnosis $\Delta = \{\psi_1, \ldots, \psi_n\}$ and some $\psi_i \in \Delta$ with δ a superformula of ψ_i, the set $\Delta' = (\Delta \setminus \{\psi_j | \psi_j \in \Delta$ and δ is a superformula of $\psi_j\}) \cup \{\delta\}$ is a diagnosis as well.*

Proof. According to Proposition 1, a superformula δ of some ψ_i can resolve (hit) at least those conflicts that ψ_i can resolve. Thus replacing ψ_i with δ in some minimal diagnosis Δ constructs a set that can still resolve all conflicts. However, in order to derive a formal diagnosis (that per definition is subset-minimal), we have to remove all subformulae of δ from Δ, which, according to Proposition 1, is not a problem regarding resolved conflicts. □

This can help in the search space exploration, as approved hypotheses (diagnoses, or in the context of Reiter's algorithm consistent sets $h(n)$) might be converted easily into multiple ones. For the HS-DAG algorithm, for instance, we derive in Section 4 a corresponding strategy for expanding a node, labeling also a consistent node's siblings as consistent (without an explicit consistency check) if their edge labels refer to superformulae of the subformula at hand.

The following corollary describes the reasoning in the opposite direction; adding or replacing some ψ_i in Δ with one of its subformulae obviously cannot grow the set of C_is hit.

Corollary 1 (Infer-down). *For some set $\Delta = \{\psi_1, \ldots, \psi_n\}$ such that $SD \cup OBS \cup \{\neg AB(c_i) | c_i \in COMP \setminus \Delta\}$ is inconsistent, and a* subformula *δ of some $\psi_i \in \Delta$, the set $\Delta' = (\Delta \setminus \psi_i) \cup \{\delta\}$ is inconsistent as well.*

Considering this corollary, an HS-DAG strategy similar to the one above could be fathomed, inferring the inconsistency of a DAG node. For HS-DAG, the effects however would be hardly noticeable, due to the conflict cache that we consider standard in today's implementations. Retrieving a set not hit by $h(n) = \Delta'$ from the cache would always succeed, as an adequate one would have been registered previously for Δ (otherwise Proposition 2 would be violated). Thus we would not save an *expensive* theorem prover call.

The following variant of Corollary 1, where we do not replace ψ_i by subformula δ, but add δ to Δ, while valid for the same reasons, does allow us to prune the search space regarding subformulae.

Lemma 2 (Prune-down). *For some set $\Delta = \{\psi_1, \ldots, \psi_n\}$ such that $SD \cup OBS \cup \{\neg AB(c_i) | c_i \in COMP \setminus \Delta\}$ is inconsistent, and a* subformula *δ of some $\psi_i \in \Delta$, the set $\Delta' = \Delta \cup \delta$ is inconsistent. Furthermore Δ' and any of its supersets cannot be a diagnosis.*

Proof. Obviously, Δ' hits exactly those conflicts C_i also hit by Δ (according to Proposition 2), so that $SD \cup OBS \cup \{\neg AB(c_i) | c_i \in COMP \setminus \Delta'\}$ cannot be consistent. Furthermore, by Proposition 1, ψ_i hits (at least) all the conflicts that δ hits, so that one could remove δ from Δ' and any of its supersets without affecting the set of conflict sets hit by them respectively. Thus neither Δ' nor any of its supersets can be a diagnosis that, per Def. 5, has to be subset-minimal. $\square$

Regarding this lemma, we would like to note that HS-DAG perfectly implements this reasoning. That is, when retrieving a label for some non-leaf node n, it asks for some C_i not hit by $h(n)$. A corresponding C_i cannot contain a subformula of some $\psi_i \in h(n)$ due to Proposition 2.

Interestingly enough, the specific definition of a diagnosis allows us to consider some aspects of this reasoning also for superformulae, so that we can prune the search space also regarding superformulae.

Lemma 3 (Prune-up)**.** *For some $\Delta = \{\psi_1, \ldots, \psi_n\}$ such that $SD \cup OBS \cup \{\neg AB(c_i) | c_i \in COMP \setminus \Delta\}$ is inconsistent, adding some δ ($\Delta' = \Delta \cup \{\delta\}$) that is a superformula of some $\psi_i \in \Delta$ cannot yield a diagnosis.*

Proof. By Proposition 1 we know that δ hits (at least) all the conflicts that ψ_i hits, so that one could remove ψ_i from Δ' without affecting the set of hit conflict sets. As diagnoses have to be subset-minimal according to Definition 5, Δ' cannot be a diagnosis then. Obviously, only adding elements to Δ' cannot resolve the issue at hand, so that also no superset of Δ' can be a diagnosis. $\square$

The effect of this lemma is that when some part of a solution is established (e.g. during a structured conflict-driven search with HS-DAG, or for a partial assignment when computing diagnoses directly with a SAT-solver), we can rule out all superformulae of any ψ_i already considered up to the point where we remove ψ_i again (e.g. during some backtracking step in the SAT-solver).

Summarizing, our lemmas allow us to dynamically adapt and focus the search for diagnoses with easily derived positive or negative data. In the next section we discuss the adoption of our reasoning in the context of the well-known HS-DAG algorithm.

4 HS-DAG

For the HS-DAG algorithm we implemented the reasoning from Lemmas 1 and 3 in the procedures INFERUP and PRUNEUP covered by Algorithms 2 and 1, respectively. Please note that HS-DAG covers the reasoning from Lemma 2 by construction, as mentioned before.

Prior to the expansion process of an (inconsistent) HS-DAG node, we call the procedure PRUNEUP. It discards from n's intended label the superformulae of all $\psi_i \in h(n)$. As conflicts may comprise multiple (overlapping) "chains" to the parse tree's root, we mark those superformulae in $\mathcal{T}$ which have been examined already.

The label $\ell(n)$ of a node n is either a conflict, "✓" (consistent), "×" (closed) or "(yet) undefined". We assume now that HS-DAG expands an inconsistent node by iterating over its label/conflict C, considering first those subformulae farthest from v_φ in the parse tree $\mathcal{T}$.

Whenever a node n is found to be consistent (i.e. $h(n)$ is "consistent"), under certain conditions INFERUP labels siblings whose incoming edges are labeled with superformulae with ✓ too. In lines 7 to 8 we make the corresponding HS-DAG subset check whether there is a subset in $h(n')$ that is a diagnosis, checking whether n' should be closed. Again, we stop following a path to the parse tree's root whenever we encounter a superformula already considered.

Algorithm 1 Pruning conflict sets in HS-DAG.

Requires: n — HS-DAG node being expanded

procedure PRUNEUP(n, $\mathcal{T}$):
    CLEARMARKS($\mathcal{T}$)
    $C \leftarrow \ell(n)$
    for all $\psi \in h(n)$ **do**
        while $\psi \neq$ NULL $\wedge$ ψ not marked **do**
            $\psi \leftarrow$ PARENT($\psi, \mathcal{T}$)
            $C \leftarrow C \setminus \{\psi\}$
            mark ψ
    return C

Algorithm 2 Inferring new diagnoses in HS-DAG.

Requires: n — consistent HS-DAG node
Requires: ψ — subformula that led to n

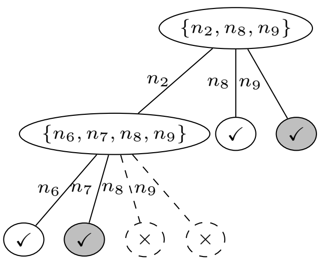
1: **procedure** INFERUP(n, $\mathcal{T}$, ψ):
2:     $C \leftarrow \ell$(PARENT(n))   ▷ parent node's conflict
3:     $\delta \leftarrow$ PARENT($\psi, \mathcal{T}$)   ▷ parent in the parse tree
4:     **while** $\delta \neq$ NULL **do**
5:         **if** δ is not marked $\wedge$ $\delta \in C$ **then**
6:             $n' \leftarrow$ GETSIBLING($(h(n) \setminus \psi) \cup \{\delta\}$)
7:             **if** $\exists m$ **s.t.** $h(m) \subseteq h(n') \wedge \ell(m) =$ ✓ **then**
8:                 $\ell(n') \leftarrow \times$
9:             **else**
10:                 $\ell(n') \leftarrow$ ✓
11:             mark δ
12:         **else**
13:             **break**
14:         $\delta \leftarrow$ PARENT($\delta, \mathcal{T}$)

The effects of our reasoning become evident in the following two examples. Our first example is adopted from [Pill *et al.*, 2006] and was also diagnosed in [Pill and Quaritsch, 2013]. It features a two line arbiter with 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 the faulty $R_4 : \forall i \in \{1,2\} : \mathsf{G}\,(g_i \rightarrow \mathsf{X}\,(\neg g_i \,\mathsf{U}\, r_i))$ preventing additional grants until new incoming requests. Testing her specification, a designer defines an unexpectedly failing witness (i.e. a trace that should satisfy the specification but violates it) $\tau = \tau_0\tau_1(\bot)^\omega$ featuring consecutive (and instantly granted) single requests for both lines:

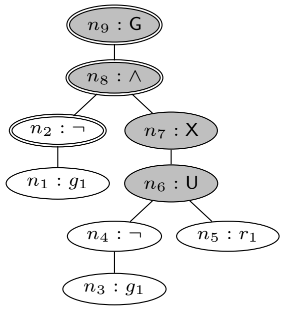
$$\tau_0 = \quad r_1 \wedge \quad g_1 \wedge \neg r_2 \wedge \neg g_2$$
$$\tau_1 = \neg r_1 \wedge \neg g_1 \wedge \quad r_2 \wedge \quad g_2$$

As already pointed out in [Pill *et al.*, 2006], the problem in this specification 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 requests that are not provided by τ, and which is presumably not in the designer's intent.

Our standard HS-DAG implementation, as used also in [Pill and Quaritsch, 2013], obtained for this scenario 31



(a) HS-DAG tree, inferred nodes shaded, pruned "edges" dashed.



(b) Parse tree, the two conflicts are shaded/double lined.

Figure 1: HS-DAG run for the arbiter formula.

diagnoses, including the one pinpointing to wrong *until* operators in both instances of R_4. It issued 34 theorem prover (SAT solver) calls in total, when building its DAG comprising 44 nodes (cf. Table 2).

The effects of our optimizations can be seen in Table 2. While the number of nodes constructed by HS-DAG could be reduced from 44 to 38 (some minimum number of nodes is needed to represent the 31 diagnoses) using PRUNEUP, the number of consistency checks that require a theorem prover call could be cut down from 31 to 13 (-58%) via 18 diagnoses inferred using INFERUP. This resulted also in a run-time reduction (over 100 runs) of more than 46% even for this simple example. Using the PRUNEUP node-reduction alone resulted in a slight but negligible (<1%) run-time penalty. Using INFERUP and PRUNEUP aggregates the advantages, offering fewest nodes as well as an attractive run-time.

For visualizing the effects of our INFERUP and PRUNEUP optimizations, we extract an even smaller example from the arbiter requirement R_4. As PRUNEUP only affects DAG levels with $|h(n)| > 1$, we purposefully inject a second fault in R_4 by replacing the logic OR (in the rewritten implication) with a logic AND: $\varphi = \mathsf{G}\,(\neg g_1 \wedge \mathsf{X}\,(\neg g_1\,\mathsf{U}\,r_1))$ and consider a single line. The trace consists of a single request and grant: $\tau = \tau_0(\bot)^\omega$ with $\tau_0 = r_1 \wedge g_1$. Figure 1 depicts the DAG and parse tree for this example.

We start expanding the root node using n_2 (inconsistent) and n_8. As $\{n_8\}$ is consistent, we can infer $\{n_9\}$ to be consistent as well, as n_9 is a superformula of n_8 (see Figure 1b). For the node labeled $\{n_6, n_7, n_8, n_9\}$, we can skip n_8 and n_9 in the expansion as their subtrees generate supersets of $\{n_8\}$ and $\{n_9\}$ only (dashed edges/nodes). Similar to the level above, we can infer $\{n_2, n_7\}$ to be a diagnosis due to $\{n_2, n_6\}$ being a diagnosis and n_7 a superformula of n_6.

	–	P↑	I↑	P↑ + I↑
# HS-DAG nodes	44	38	44	38
# TP consistency checks	31	31	13	13
# TP conflict computations	3	3	3	3
# pruned "edges"	–	6	–	6
# inferred diagnoses	–	–	18	18
# diagnoses	31	31	31	31
run-time (sec.)	0.3801	0.3821	0.2029	0.2033

Table 2: HS-DAG statistics for the arbiter example using no/PRUNEUP/INFERUP/PRUNEUP+INFERUP optimization

4.1 Experimental Results

We applied our optimizations to the Python (CPython 2.7.1) implementation used in [Pill and Quaritsch, 2013]. 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 test samples, we generated random LTL formulae as suggested in [Daniele *et al.*, 1999] with $N = \lfloor|\varphi|/3\rfloor$ variables and a uniform distribution of LTL operators. We injected *triple* faults in order to derive φ_m from φ, and using our LTL encoding, we derived an assignment for $\tau \wedge \varphi \wedge \neg\varphi_m$ that defines τ for $k = 100$ and $l = 50$. We verified that φ_m is a valid diagnosis considering φ and τ.

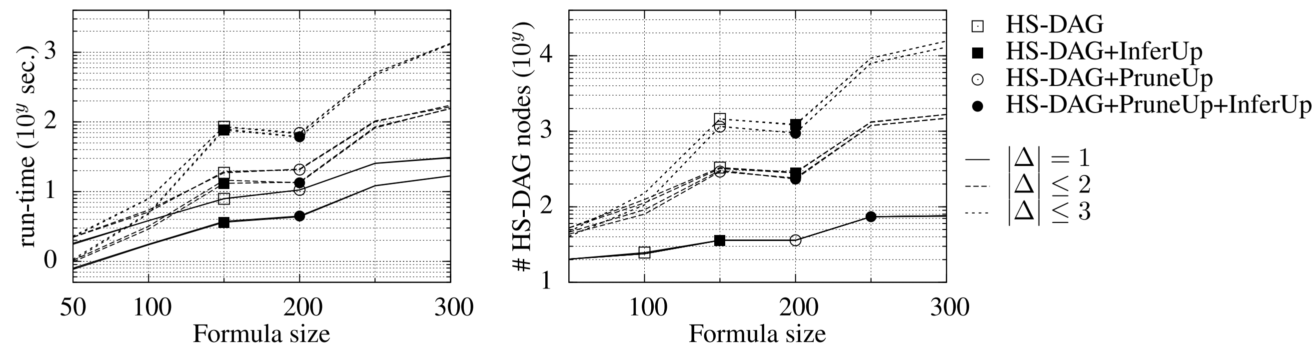
To obtain the results in Figure 2, we generated 10 random diagnosis problems as outlined above for any $|\varphi|$ in $\{50, 100, \ldots, 300\}$, ran HS-DAG ten times with a diagnosis cardinality limit of 1, 2 and 3 (single, double and triple faults) with our various optimizations applied, and plotted average values. For the single fault diagnosis runs (solid lines), we observe a run-time reduction of up to approx. 60% due to INFERUP. The run-time benefit diminishes with rising maximum diagnosis cardinality, when, intuitively, the number of diagnoses (and thus inferable nodes) grows slower than the total number of DAG nodes. While PRUNEUP shows virtually no influence on the run-time, Figure 2b depicts its impact on the number of DAG nodes constructed for a specific problem. Growing with rising maximum diagnosis cardinality, a reduction of up to 23% was possible for $|\varphi| = 200$ and $|\Delta| \leq 3$. We thus argue that PRUNEUP eliminates HS-DAG nodes that would have been closed otherwise by subset-checks later-on.

Summarizing, while PRUNEUP could achieve a significant DAG node reduction for large diagnosis cardinalities only (i.e., in unbounded runs), INFERUP could substantially reduce run-times for the more practical, low-bound case.

5 Discussion and Conclusions

With us focusing on LTL specification diagnosis, similar ideas have been exploited previously for other domains. For example, in the context of circuit diagnosis, the concepts of dominators and cones are exploited for circuit abstraction and a diagnosis speed up. Originating in the field of program analysis using control flow graphs [Prosser, 1959; Lengauer and Tarjan, 1979] and later adopted for the analysis of digital circuits [Kirkland and Mercer, 1987], a dominating component can "overrule" the dominated ones (referred to as its corresponding *cone*) because, e.g., it is "closer to the output". As dominators for an arbitrary graph structure can be computed in linear time [Buchsbaum *et al.*, 2008], approaches such as [Siddiqi and Huang, 2007; Metodi *et al.*, 2012] focus their diagnostic search on those gates first. The resulting *top level diagnoses* are then refined by creating further potential diagnoses with dominators replaced by gates from their cone.

While cones are not directly exploitable for specification diagnosis (we would get a single (maximal) cone if applied to a static LTL parse tree or raise complexity unnecessarily when temporally unfolding it) [Mangassarian *et al.*, 2011; Le *et al.*, 2012] peruse the notion of (reverse) dominance for their SAT-based RTL debugging, resulting in implied (non-)solutions. We showed that for consistency-based diagnosis using HS-DAG, we can achieve a speed-up of about



(a) Run-time

(b) Number of HS-DAG nodes

Figure 2: Diagnosis performance for random samples.

factor two also for our problem domain, using implied (inferred) solutions. On the other hand, we showed that the implication of non-solutions does not speed up HS-DAG's diagnosis process due to the usage of a conflict set cache. Instead, we could optimize HS-DAG's search strategy in the context of a domination relation by pruning the conflicts depending on the current tree context. The latter resulted in up to 23% fewer HS-DAG nodes for our tests.

We expect our reasoning to be attractive also for similar, (temporal) formula-based descriptions. Future work will include the transfer of our search strategy optimizations to the direct diagnosis computation with SAT/constraint solvers.

Acknowledgements

This work was supported by the Austrian Science Fund (FWF): P22959-N23 ("MoDiaForTeD"). The authors would like to thank Franz Wotawa for fruitful discussions and the anonymous reviewers for their valuable comments.

References

[Buchsbaum *et al.*, 2008] A. Buchsbaum, L. Georgiadis, H. Kaplan, A. Rogers, R. Tarjan, and J. Westbrook. Linear-time algorithms for dominators and other path-evaluation problems. *SIAM Journal on Computing*, 38(4):1533–1573, 2008.

[Daniele *et al.*, 1999] M. Daniele, F. Giunchiglia, and M. Vardi. Improved automata generation for Linear Temporal Logic. In *Computer Aided Verification*, pages 249–260, 1999.

[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.

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

[Kirkland and Mercer, 1987] T. Kirkland and M. R. Mercer. A topological search algorithm for ATPG. In *ACM/IEEE Design Automation Conference*, pages 502–508, 1987.

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

[Le *et al.*, 2012] B. Le, H. Mangassarian, B. Keng, and A. Veneris. Non-solution implications using reverse domination in a modern SAT-based debugging environment. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 629–634, 2012.

[Lengauer and Tarjan, 1979] T. Lengauer and R. E. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Transactions on Programming Languages and Systems*, 1(1):121–141, 1979.

[Mangassarian *et al.*, 2011] H. Mangassarian, A. Veneris, D. E. Smith, and S. Safarpour. Debugging with dominance: On-the-fly RTL debug solution implications. In *International Conference on Computer-Aided Design*, pages 587–594, 2011.

[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.

[Pill and Quaritsch, 2013] I. Pill and T. Quaritsch. Behavioral diagnosis of LTL specifications at operator level. In *International Joint Conference on Artificial Intelligence*, pages 1053–1059, 2013.

[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 *18th Annual Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[Prosser, 1959] R. T. Prosser. Applications of Boolean matrices to the analysis of flow diagrams. In *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*, pages 133–138. ACM, 1959.

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

[Siddiqi and Huang, 2007] S. Siddiqi and J. Huang. Hierarchical diagnosis of multiple faults. In *International Joint Conference on Artificial Intelligence*, pages 581–586, 2007.

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