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 rework of rework, these figures illustrate to the high demand for tools and means that help us in getting our specifications right.

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 context 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 \( \delta \) and \( \varphi \) LTL formulae. Then an LTL formula is defined inductively as follows [Pnueli, 1977]:

- for any \( p \in AP \), \( p \) is an LTL formula
- \( \neg \varphi \), \( \varphi \land \delta \), \( \varphi \lor \delta \), \( X \varphi \), and \( \varphi \cup \delta \) are LTL formulae

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

- \( V_\varphi \) is the set of vertices such that for each subformula \( \psi \) in \( \varphi \) there is exactly one vertex \( (v_\psi) \) labeled with \( \psi \),
- \( l(v) \) is a labeling function for vertices \( v \in V_\varphi \) s.t. \( l(v_\psi) = \psi \).
\[ \varphi \in V \] is the single root vertex.

and \( E \) is \( T \)'s set of edges, s.t. for \( v_{i_0}, v_{i_2} \in V, e = (v_{i_1}, v_{i_2}) \) is in \( E \), if \( v_{i_2} \) is an operand of \( v_{i_1} \).

Note that we consider multiple occurrences of a syntactic construct to be multiple subformulas. With \( \top \) denoting logic True/High/1 and \( \bot \) denoting logic False/Low/0, the popular operators \( \delta \sigma_r, \mathcal{F}_r, \mathcal{G}_r \), and \( \delta \wedge \sigma_r \) are syntactic sugar for common formulas \( \neg(\neg \delta \sigma_r) \), \( \top \sigma_r \), \( \bot \sigma_r \), and \( \delta \sigma_r \vee \gamma \delta \sigma_r \) 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 SAT.

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 \( \tau \) is an infinite word over letters from some alphabet \( \Sigma \) of the form \( \tau = (\tau_0 \tau_1 \ldots \tau_{l-1}) \tau_l \tau_{l+1} \cdots \tau_m \) with \( k, i, \pi, \tau \in \Sigma \) for any \( 0 \leq i \leq k \) and \( \cdots \) denoting infinite repetition of the corresponding (sub-)sequence. With \( \tau^i \), we refer to \( \tau \)'s suffix starting with \( \tau_i \).

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

**Definition 4.** Given a trace \( \tau \) and an LTL formula \( \varphi \), \( \tau^i \varphi \) satisfies \( \varphi \), denoted as \( \tau^i \models \varphi \), under the following conditions:

\[
\begin{align*}
\tau^0 \models p & \iff p \in \tau, \\
\tau^i \models \neg \varphi & \iff \tau^i \not\models \varphi, \\
\tau^i \models \varphi \land \sigma & \iff \tau^i \models \varphi \land \tau^i \models \sigma, \\
\tau^i \models \varphi \lor \sigma & \iff \tau^i \models \varphi \lor \tau^i \models \sigma, \\
\tau^i \models X \varphi & \iff \tau^{i+1} \models \varphi, \\
\tau^i \models \varphi \Rightarrow \sigma & \iff \forall j \leq m < j \tau^m \models \varphi.
\end{align*}
\]

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 \( \text{COMP} \), assumption predicates \( \neg \mathcal{A} \mathcal{B}(c_i) \) for all \( c_i \in \text{COMP} \) encoding whether \( c_i \) behaves abnormally, a system description \( \mathcal{S} \) defining the correct behavior \( \neg \mathcal{A} \mathcal{B}(c_i) \Rightarrow \text{NominalBehavior}(c_i) \), and some actual observations \( OBS \) about a system’s behavior, the system is considered to be at fault iff \( \text{SD} \cup \text{OBS} \cup \{\neg \mathcal{A} \mathcal{B}(c_i) \mid c_i \in \text{COMP} \} \) is inconsistent. While a minterm in the assumptions defines a specific state of the system, a diagnosis \( \Delta \) 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 \( (\text{SD}, \text{COMP}, \text{OBS}) \) is a subset-minimal set \( \Delta \subseteq \text{COMP} \) such that \( \text{SD} \cup \text{OBS} \cup \{\neg \mathcal{A} \mathcal{B}(c_i) \mid c_i \in \text{COMP} \setminus \Delta \} \) is consistent.

Reiter proposes to compute the set of diagnoses as the minimal hitting sets of the set \( \text{CS} \) of (not necessarily minimal) conflicts for \((\text{SD}, \text{COMP}, \text{OBS})\).
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 to 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 ψι are freed in that their values become undefined. Via the individual subformulae’s encodings, the chosen values for ψι however still influence (depending on the operators) the evaluation of the variables of its superformulae (those formulae on the path from root vertex ϕ to ψι). Thus, if there is some satisfying assignment for ψι when considering ϕ faulty, this assignment is still a satisfying one when considering a superformula δ to be at fault, and freeing the assignments of δ’s subformulae (not signals!) that are irrelevant for the evaluation of ϕ anyway (due to $AB(δ)$).

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

<table>
<thead>
<tr>
<th>ϕ</th>
<th>Unfolding rationales</th>
<th>I</th>
<th>Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T/\perp$ ϕι ↔ $T/\perp$</td>
<td>✓</td>
<td>(a) ϕι/ϕι</td>
<td></td>
</tr>
<tr>
<td>$\delta \land \varphi_i$ ↔ $(\delta_1 \land \varphi_i)$</td>
<td>✓</td>
<td>(b1) $\varphi_i \lor \delta_1$</td>
<td>✓</td>
</tr>
<tr>
<td>$\delta \lor \varphi_i$ ↔ $(\delta_1 \lor \varphi_i)$</td>
<td>✓</td>
<td>(b2) $\varphi_i \lor \delta_1$</td>
<td>✓</td>
</tr>
<tr>
<td>$\delta \lor \varphi_i$ ↔ $(\delta_1 \lor \varphi_i)$</td>
<td>✓</td>
<td>(b3) $\varphi_i \lor \delta_1 \lor \delta_1$</td>
<td>✓</td>
</tr>
<tr>
<td>$-\delta \land \varphi_i$ ↔ $-\delta_1$</td>
<td>✓</td>
<td>(d1) $\varphi_i \lor \delta_1$</td>
<td>✓</td>
</tr>
<tr>
<td>$\delta_1 \land \varphi_{i+1}$</td>
<td>✓</td>
<td>(d2) $\varphi_i \lor \delta_1$</td>
<td>✓</td>
</tr>
<tr>
<td>$\delta_1 \land \varphi_{i+1} \rightarrow \varphi_i$</td>
<td>✓</td>
<td>(e1) $\varphi_i \lor \delta_1$</td>
<td>✓</td>
</tr>
<tr>
<td>$\delta_1 \land \varphi_{i+1} \rightarrow \varphi_i$</td>
<td>✓</td>
<td>(e2) $\varphi_i \lor \delta_1$</td>
<td>✓</td>
</tr>
<tr>
<td>$\delta \lor \varphi_i$</td>
<td>✓</td>
<td>(f1) $\varphi_i \lor \delta_1$</td>
<td>✓</td>
</tr>
<tr>
<td>$\delta \lor \varphi_i$</td>
<td>✓</td>
<td>(f2) $\varphi_i \lor \delta_1$</td>
<td>✓</td>
</tr>
<tr>
<td>$\delta \lor \varphi_i$</td>
<td>✓</td>
<td>(g) $\varphi_i \lor \delta_1$</td>
<td>✓</td>
</tr>
<tr>
<td>$\delta_1 \lor \varphi_{i+1}$</td>
<td>✓</td>
<td>(h) $\delta_1 \lor \varphi_{i+1}$</td>
<td>✓</td>
</tr>
<tr>
<td>$\delta_1 \lor \varphi_{i+1}$</td>
<td>✓</td>
<td>(i) $\varphi_i \lor \delta_1 \lor \delta_1$</td>
<td>✓</td>
</tr>
</tbody>
</table>

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 $0 \leq i \leq k$.

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 $C_i$ 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 $\Delta'$ we can derive further diagnoses $\Delta'$.)

Lemma 1 (Infer-up). For a diagnosis $\Delta = \{ψ_1, ..., ψ_n\}$ and some $ψ_i ∈ Δ$ with $δ$ a superformula of $ψ_i$, the set $Δ′ = (Δ \setminus \{ψ_i\} ∪ δ) \lor \{δ\}$ 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 $C_i$’s hit.

Corollary 1 (Infer-down). For some set $Δ = \{ψ_1, ..., ψ_n\}$ such that $SD \lor OBS \lor (¬AB(ψ_i)) ∈ COMP \setminus Δ$ is inconsistent, and a subformula $δ$ of some $ψ_i ∈ $ $Δ$, the set $Δ′ = \{Δ \setminus \{ψ_i\} \lor \{δ\}\}$ is inconsistent as well.

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 $C_i$’s hit.

Lemma 2 (Prune-down). For some set $Δ = \{ψ_1, ..., ψ_n\}$ such that $SD \lor OBS \lor (¬AB(ψ_i)) ∈ COMP \setminus Δ$ is inconsistent, and a subformula $δ$ of some $ψ_i ∈ $ $Δ$, the set $Δ′ = \{Δ \setminus \{ψ_i\} \lor \{δ\}\}$ is inconsistent as well. Furthermore $Δ′$ and any of its supersets cannot be a diagnosis.

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.

Corollary 1 (Infer-down). For some set $Δ = \{ψ_1, ..., ψ_n\}$ such that $SD \lor OBS \lor (¬AB(ψ_i)) ∈ COMP \setminus \{δ\}$ is inconsistent, and a subformula $δ$ of some $ψ_i ∈ Δ$, the set $Δ′ = \{Δ \setminus \{ψ_i\} \lor \{δ\}\}$ is inconsistent as well.

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.
Proof. Obviously, $\Delta'$ hits exactly those conflicts $C_i$ also hit by $\Delta$ (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, $\psi_i$ hits (at least) all the conflicts that $\delta$ hits, so that one could remove $\delta$ from $\Delta'$ and any of its supersets without affecting the set of hit conflicts hit by them. Thus neither $\Delta'$ nor any of its supersets can be a diagnosis that, per Def. 5, has to be subset-minimal. □

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 $n_i$ it asks for some $C_i$ not hit by $h(n_i)$. 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 $\delta (\Delta' = \Delta \cup \{\delta\})$ that is a superformula of some $\psi_i \in \Delta$ cannot yield a diagnosis.

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

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 $\psi_i$ already considered up to the point where we remove $\psi_i$ 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 PRUNEUp 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 $T$ which have been examined already.

The label $\ell(n)$ of a node $n$ is either a conflict, “$\psi$” (consistent), “$\times$” (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 $v_\gamma$ in the parse tree $T$.

Whenever a node $n$ is found to be consistent (i.e. $h(n)$ is “consistent”), under certain conditions PRUNEUp labels siblings whose incoming edges are labeled with superformulae

```plaintext
Algorithm 1 Pruning conflict sets in HS-DAG.

Requires: $n$ — HS-DAG node being expanded
1: procedure PRUNEUp(n, T):
2: CLEARMARKS(T)
3: $C \leftarrow \ell(n)$
4: for all $\psi \in h(n)$ do
5: while $\psi \neq NULL$ and $\psi$ not marked do
6: $\psi \leftarrow PARENT(\psi, T)$
7: $C \leftarrow C \setminus \{\psi\}$
8: mark $\psi$
9: return $C$
```

Algorithm 2 Inferring new diagnoses in HS-DAG.

Requires: $n$ — consistent HS-DAG node

Requires: $\psi$ — subformula that led to $n$

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

with $\sqrt{\tau}$ 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.

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 Quartitsch, 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 $R_4$ : $\forall i \in \{1, 2\} : G(\neg g_i \rightarrow X(\neg g_i \cup \neg g_\gamma) \setminus (-g_i \cup -g_\gamma)$ 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): $\tau = \tau_0 \tau_1 \tau_2 \tau_3$ featuring consecutive (and instantly granted) single requests for both lines:

$\tau_0 = r_1 \land g_1 \land r_2 \land \neg g_2$

As already pointed out in [Pill et al., 2006], the problem in this specification is the until operator $\neg g_i \cup r_i$ in $R_4$ that should be replaced by its weak version $\neg g_i \cup r_i$ 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 $\tau$, and which is presumably not in the designer’s intent.

Our standard HS-DAG implementation, as used also in [Pill and Quartitsch, 2013], obtained for this scenario 31
diagnoses, including the one pinpointing to wrong until operators in both instances of $R_4$. It issued 34 theorem prove (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 PRUNEP, 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 INFERU. This resulted also in a run-time reduction (over 100 runs) of more than 46% even for this simple example. Using the PRUNEP node-reduction alone resulted in a slight but negligible (<1%) run-time penalty. Using INFERU and PRUNEP aggregates the advantages, offering fewest nodes as well as an attractive run-time.

For visualizing the effects of our INFERU and PRUNEP optimizations, we extract an even smaller example from the optimization, we extract an even smaller example from the run-time.

We applied our optimizations to the Python (CPython 2.7.1) implementation used in [Pill and Quartzs, 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 $N = |\phi|/3$ variables and a uniform distribution of LTL operators. We injected triple faults in order to derive $\phi_m$ from $\phi$, and using our LTL encoding, we derived an assignment for $\tau \land \phi \land \neg \phi_m$ that defines $\tau$ for $k = 100$ and $l = 50$. We verified that $\phi_m$ is a valid diagnosis considering $\phi$ and $\tau$.

To obtain the results in Figure 2, we generated 10 random diagnosis problems as outlined above for any $|\phi|$ in {50, 100, …, 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 INFERU. 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 INFERU 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 $|\phi| = 200$ and $|\Delta| \leq 3$. We thus argue that PRUNEP eliminates HS-DAG nodes that would have been closed otherwise by subset-checks later-on.

Summarizing, while PRUNEP could achieve a significant DAG node reduction for large diagnosis cardinalities only (i.e., in unbounded runs), INFERU 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-)isolations. We showed that for consistency-based diagnosis using HS-DAG, we can achieve a speed-up of about

![Figure 1: HS-DAG run for the arbiter formula.](image)

<table>
<thead>
<tr>
<th># HS-DAG nodes</th>
<th>$P^\uparrow$</th>
<th>$I^\uparrow$</th>
<th>$P^\uparrow + I^\uparrow$</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td>31</td>
<td>3</td>
<td>33</td>
</tr>
<tr>
<td>31</td>
<td>13</td>
<td>13</td>
<td>26</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>18</td>
<td>18</td>
<td>36</td>
</tr>
<tr>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
</tr>
</tbody>
</table>

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


