Trace Diagnostics for Signal-based Temporal Properties

Chaima Boufaied, Claudio Menghi, *Member, IEEE,* Domenico Bianculli, *Member, IEEE,* and Lionel C. Briand, *Fellow, IEEE*

Abstract—

Trace checking is a verification technique widely used in Cyber-physical system (CPS) development, to verify whether execution traces satisfy or violate properties expressing system requirements. Often these properties characterize complex signal behaviors and are defined using domain-specific languages, such as SB-TemPsy-DSL, a pattern-based specification language for signal-based temporal properties. Most of the trace-checking tools only yield a Boolean verdict. However, when a property is violated by a trace, engineers usually inspect the trace to understand the cause of the violation; such manual diagnostic is time-consuming and error-prone. Existing approaches that complement trace-checking tools with diagnostic capabilities either produce low-level explanations that are hardly comprehensible by engineers or do not support complex signal-based temporal properties.

In this paper, we propose *TD-SB-TemPsy*, a trace-diagnostic approach for properties expressed using SB-TemPsy-DSL. Given a property and a trace that violates the property, *TD-SB-TemPsy* determines the root cause of the property violation. *TD-SB-TemPsy* relies on the concepts of *violation cause*, which characterizes one of the behaviors of the system that may lead to a property violation, and *diagnoses*, which are associated with violation causes and provide additional information to help engineers understand the violation cause. As part of *TD-SB-TemPsy*, we propose a language-agnostic methodology to define violation causes and diagnoses. In our context, its application resulted in a catalog of 34 violation causes, each associated with one diagnosis, tailored to properties expressed in SB-TemPsy-DSL.

We assessed the applicability of *TD-SB-TemPsy* on two datasets, including one based on a complex industrial case study. The results show that *TD-SB-TemPsy* could finish within a timeout of $1\,\mathrm{min}$ for $\approx 83.66\%$ of the trace-property combinations in the industrial dataset, yielding a diagnosis in $\approx 99.84\%$ of these cases; moreover, it also yielded a diagnosis for all the trace-property combinations in the other dataset. These results suggest that our tool is applicable and efficient in most cases.

Index Terms—Diagnostics, Trace checking, Run-time verification, Temporal properties, Specification patterns, Cyber-physical systems, Signals

1 Introduction

Cyber-physical system (CPS) development requires engineers to verify whether the system meets its requirements. In industrial contexts, verification is often performed through *trace-checking* tools (e.g., [1, 2, 3, 4, 5, 6]). Engineers collect traces, sequences of records representing the behavior of the system, and use trace-checking tools to check whether the traces satisfy or violate properties expressing the system requirements. If properties are violated, the system has faults that need to be identified and corrected.

In the case of *pattern-based* trace-checking tools, properties are expressed using pattern-based languages. Pattern-based languages contain domain-specific constructs to express complex requirements [7] that increase their usability in industrial contexts [8, 7]. In this work, we consider requirements expressed in SB-TemPsy-DSL [1], a pattern-based language that can express complex signal behaviors based on a recent taxonomy [9]. This language enables engineers to write properties describing important types of requirements for industrial CPSs, through constructs that express complex signal behaviors, such as spikes and oscillations.

When a property is checked on a trace, trace-checking tools usually provide a *Boolean verdict*: *true* if the trace satisfies the property, *false* otherwise. When the property is violated by a trace, engineers usually inspect the trace to understand the cause of the violation, leading to the analysis of a high number of records. For example, in our industrial case study in the satellite domain, the average number of records included in 361 traces is 438224. Inspecting a large number of records, and checking the causes of property violations, requires in general significant time. Additionally, this activity is error-prone and engineers may fail to identify the actual cause of the property violation. Therefore, they need automated tools that can explain the reasons leading to the violation of the properties. These tools should provide

- *C. Boufaied is with the school of EECS, University of Ottawa, Ottawa, ON K1N 6N5, Canada (e-mail:chaima.boufaied@uottawa.ca). Part of this work was done when she was affiliated with the Interdisciplinary Centre for Security, Reliability, and Trust (SnT) of the University of Luxembourg.*
- *C. Menghi is with University of Bergamo, Bergamo, Italy and McMaster University, Hamilton, Canada (e-mail: claudio.menghi@unibg.it). Part of this work was done when he was affiliated with the Interdisciplinary Centre for Security, Reliability, and Trust (SnT) of the University of Luxembourg.*
- *D. Bianculli is with the Interdisciplinary Centre for Security, Reliability, and Trust (SnT) of the University of Luxembourg, Luxembourg (e-mail: domenico.bianculli@uni.lu).*
- *L. Briand holds shared appointments with the Interdisciplinary Centre for Security, Reliability, and Trust (SnT) of the University of Luxembourg, Luxembourg and the school of EECS, University of Ottawa, Ottawa, ON K1N 6N5, Canada (e-mail: lionel.briand@uni.lu).*

diagnostic information enabling engineers to understand the cause of violations.

Two complementary strategies were proposed in the literature to help engineers in these activities: (i) isolating *slices of traces* that explain the property violation; and (ii) checking whether traces show common *behaviors that lead to the property violation*. These two complementary strategies are discussed in the following.

Approaches that isolate *slices of the trace* that explain the property violation (e.g., [10, 11, 12, 13]) usually assume that the properties are specified using a logical formula. To explain the property violation, these approaches iteratively analyze the sub-formulae of the logical formula and identify minimal slices of the trace that explain the satisfaction or violation of each sub-formula. Using this approach, the size of the explanation increases with the number of sub-formulae of the logical formula expressing the property. For properties expressed using pattern-based languages, which provide domain-specific constructs encoding complex logical formulae, using such an approach is likely to produce large explanations that are hardly comprehensible by engineers. Besides, none of these approaches was implemented and evaluated on realistic case studies.

Approaches that check for the presence of common *behaviors leading to the property violation* (e.g., [14, 15, 16]), assume that such behaviors correspond to common causes of such violation. Each cause therefore encodes one of the behaviors, observed in the trace, that may lead to a property violation and help explain it. However, existing approaches do not support complex signal-based temporal properties of CPS, such as the one expressed using SB-TemPsy-DSL. Besides, it is unclear how to extend these approaches to support signal-based temporal properties, since such approaches do not come with a precise methodology that describes how to add new causes that support more complex properties.

In this work, we propose *TD-SB-TemPsy*, a trace-diagnostic approach for signal-based temporal properties. *TD-SB-TemPsy* takes as input a trace and a property expressed using SB-TemPsy-DSL and violated by the trace; it provides as output an explanation that describes why the property is violated on that trace.

To detect the source of the property violation, we define the notions of *violation cause* and *diagnosis*. A violation cause characterizes one of the behaviors of the system that may lead to a property violation. For example, for a property requiring a signal to show a spike with an amplitude and a width lower than specific thresholds, the absence of any spike behavior in a signal is a violation cause. Diagnoses are associated with violation causes and provide additional information to help engineers understand such causes. For example, a diagnosis for the previous violation cause, for the case in which the value of the signal is increasing over time, contains two records (timestamps and signal values) where the signal shows its minimum and maximum values, while increasing. These values allow engineers to understand the range of values taken by the signal while it exhibits an increasing behavior.

We propose a novel *methodology to define violation causes and diagnoses* (Section 5). Our methodology provides formal guarantees of the soundness of the proposed violation causes: if a violation cause holds on a trace, the corresponding property is violated. Though we applied our methodology to define violation causes for properties expressed using SB-TemPsy-DSL, our methodology is language-agnostic and can therefore be applied to other pattern-based specification languages such as TemPsy [17] and FRETISH [18]. To further support this claim, we also sketch how to apply our methodology to one construct supported by the latter.

We present a *catalog of 34 violation causes, each associated with one diagnosis,* for signal-based temporal properties expressed in SB-TemPsy-DSL (Section 6). These violation causes are not complete as they do not encode all the possible reasons that may lead to a property violation, but are the results of applying our methodology in the context of our industrial case study. Indeed, such a catalogue of violation causes and diagnoses has been defined (and validated) together with a group of system and software engineers of our industrial partner, with the goal of maximizing the usefulness of a diagnosis for a certain violation cause. However, following the same methodology, users can add new violation causes depending on their specific needs or on the requirements of particular domains.

We implemented *TD-SB-TemPsy* as a plugin for SB-TemPsy-Check [1], a trace-checking tool for SB-TemPsy-DSL. We assessed the applicability of *TD-SB-TemPsy* on a large, proprietary industrial dataset from the satellite domain (PROP-SAT), as well as a smaller dataset (AFC) generated from a benchmark model used in the ARCH competition [19]. *TD-SB-TemPsy* could finish within a timeout of $1\,\mathrm{min}$ for $\approx 83.66\%$ of the trace-property combinations in the PROP-SAT dataset, yielding a diagnosis in $\approx 99.84\%$ of these cases; moreover, it also yielded a diagnosis for all the trace-property combinations in the AFC dataset.

Significance. Since diagnoses were provided in $\approx 99.84\%$ of the cases for which no timeout occurred in the PROP-SAT dataset, and in the totality of the trace-property combinations in the AFC one, *TD-SB-TemPsy* was deemed widely applicable across trace-property combinations in both datasets. Given the high expressiveness of SB-TemPsy-DSL, the many violation causes and related diagnoses we have defined in *TD-SB-TemPsy*, and the run-time performance of our tool, we expect significant impact for this technology across many CPS domains. Moreover, the methodology for defining violation causes and diagnoses can be adopted by other researchers working on the problem of trace diagnostics in the context of run-time verification.

To summarize, the main contributions of this paper are:

- *TD-SB-TemPsy*, a trace-diagnostic approach for signal-based temporal properties expressed in SB-TemPsy-DSL, based on the concepts of *violation cause* and *diagnosis*;
- a language-agnostic methodology for defining violation causes and diagnoses, with formal guarantees of the soundness of the proposed violation causes, and its application to SB-TemPsy-DSL;
- a catalog of 34 violation causes, each associated with one diagnosis, for signal-based temporal properties expressed in SB-TemPsy-DSL;
- a comprehensive evaluation of the applicability of *TD-SB-TemPsy* on two datasets, including one based on a complex industrial case study.

β	2.0	153.5	55.0	0.5	80.0	203.5	20.0	0.5
ρ	1.0	52.5	125.0	125.5	25.0	75.5	35.0	200.5
timestamp	0.0	0.2	0.9	1.8	3.0	4.9	5.7	6.0

Record r_3

Figure 1. A fragment of a trace from our case study.

Paper structure. This paper is organized as follows. Section 2 introduces our case study from the satellite domain and identifies concrete motivations for our work. Section 3 illustrates the syntax and semantics of SB-TemPsy-DSL. Section 4 presents *TD-SB-TemPsy*, our pattern-based trace-diagnostic approach. Section 5 describes our methodology to define violation causes and diagnoses. Section 6 presents the violation causes and diagnoses proposed in this work. Section 8 reports on the evaluation of the applicability of *TD-SB-TemPsy* on two datasets. Section 9 discusses the practical implications of our approach. Section 10 surveys related work. Section 11 concludes the paper, providing directions for future work.

2 Case Study and Motivations

Our case study is a satellite developed by our industrial partner. This is a representative CPS as it contains many complex software components that interact with actuators and sensors of the satellite.

During the satellite development, and after its deployment, engineers collect traces that describe the behavior of the satellite. A fragment of one of these traces is depicted in Figure 1 and plotted in Figure 2. A trace is a sequence of records that describe how the values of some signals change over time. For example, the fragment of the trace in Figure 1 contains eight records. Each record contains a timestamp, identifying the time at which the record was collected, and the values assumed by some variables, each recording the values of one of the monitored signals at that time. In the example, the variables β and ρ record respectively the signals representing the beta angle [20] and the pointing error [21] of the satellite. For example, for record r_3 the timestamp is 0.9, and the values of the variables β and ρ are respectively $55.0°$ and $125.0°$. The recording interval of a trace is the difference between the maximum and the minimum timestamps. For example, the recording interval of the trace in Figure 1 is 6 s.

After the traces are collected, engineers analyze whether the behaviors recorded in the traces satisfy the CPS requirements. An example of a requirement (inspired by the ones from the case study) is the following:

> R1: "*Within the trace, the beta angle shall contain at least one spike with an amplitude lower than* $90°$ *and a width less than* $0.5\,\text{s}$".

The beta angle is the angle between the orbital plane of the satellite and the vector of the Sun (i.e., the direction from which the Sun is shining). After deployment, the satellite aligns its orbital plane. Therefore, β shall contain a spike with an amplitude lower than $90°$. The trace shown in Figure 1 violates the requirement R1. As we will discuss in the next section, automated trace-checking tools, such as SB-TemPsy-Check [1] (see section 3), can verify whether a trace satisfies or violates a requirement. However, they do not provide any additional information to help engineers understand the cause of the violation. This means that engineers have to manually inspect the values of the variables recorded in the trace records and check why these values led to the violation of the requirement. In our example, looking at the plot in Figure 2, one can see that the two spikes (i.e., *spike1* and *spike2* defined over the time intervals $[0, 1.8]$ and $[1.8, 6.0]$) of signal β have an amplitude value ($A_1 = 153.5°$ and $A_2 = 203.5°$) greater than $90°$ and show a width ($w_1 = 1.8\,\text{s}$ and $w_2 = 4.2\,\text{s}$) greater than $0.5\,\text{s}$. Our pattern-based diagnostic approach (see section 4) aims to automatically detect the causes of requirement violation.

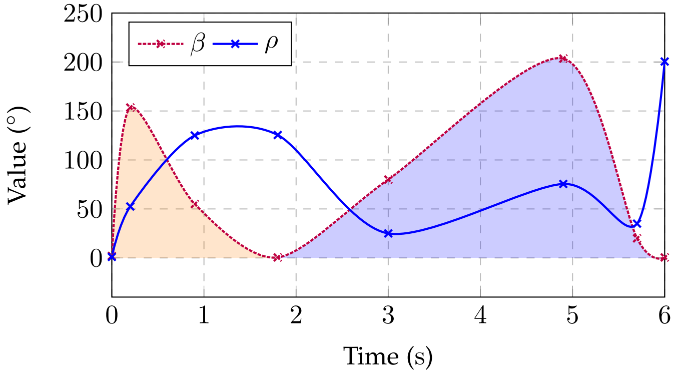


Figure 2. Graphical representation of the trace in Figure 1.

Property	$\phi ::= \ \phi_1 \ \texttt{or} \ \phi_2 \mid \delta$
Clause	$\delta ::= \ \delta_1 \ \texttt{and} \ \delta_2 \mid \alpha$
Atom	$\alpha ::= \ \texttt{not}\ \mathsf{sc} \mid \mathsf{sc}$
Scope	$\mathsf{sc} ::= \texttt{globally}\ \mathsf{p} \mid \texttt{before}\ \mathsf{t}\ \mathsf{p} \mid \texttt{after}\ \mathsf{t}\ \mathsf{p} \mid \texttt{at}\ \mathsf{t}\ \mathsf{p} \mid$ $\texttt{before}\ \mathsf{p}_1\ \mathsf{p} \mid \texttt{after}\ \mathsf{p}_1\ \mathsf{p} \mid$ $\texttt{between}\ \mathsf{t}_1\ \texttt{and}\ \mathsf{t}_2\ \mathsf{p} \mid \texttt{between}\ \mathsf{p}_1\ \texttt{and}\ \mathsf{p}_2\ \mathsf{p}$
Pattern	$\mathsf{p} ::= \ \texttt{assert}\ \mathsf{c} \mid \mathsf{s}\ \texttt{becomes} \sim \mathsf{v} \mid$ $\texttt{if}\ \mathsf{p}_1\ \texttt{then}\ [\texttt{within} \bowtie \mathsf{t}]\ \mathsf{p}_2 \mid$ $\texttt{exists spike in}\ \mathsf{s}$ $[\texttt{with}\ [\texttt{width} \ \sim_1 \mathsf{v}_1] \mid [\texttt{amplitude} \ \sim_2 \mathsf{v}_2]] \mid$ $\texttt{exist oscillation in}\ \mathsf{s}$ $[\texttt{with}\ [\texttt{p2pAmp} \ \sim_1 \mathsf{v}_1][\texttt{period} \ \sim_2 \mathsf{v}_2]] \mid$ $\mathsf{s}\ \texttt{rises}\ [\texttt{monotonically}]\ \texttt{reaching}\ \mathsf{v} \mid$ $\mathsf{s}\ \texttt{falls}\ [\texttt{monotonically}]\ \texttt{reaching}\ \mathsf{v} \mid$ $\mathsf{s}\ \texttt{overshoots}\ [\texttt{monotonically}]\ \mathsf{v}_1\ \texttt{by}\ \mathsf{v}_2 \mid$ $\mathsf{s}\ \texttt{undershoots}\ [\texttt{monotonically}]\ \mathsf{v}_1\ \texttt{by}\ \mathsf{v}_2$ $\bowtie ::= \texttt{exactly} \mid \texttt{at most} \mid \texttt{at least}$
Condition	$\mathsf{c} ::= \ \mathsf{c}_1\ \texttt{and}\ \mathsf{c}_2 \mid \mathsf{c}_1\ \texttt{or}\ \mathsf{c}_2 \mid \mathsf{s} \ \sim \mathsf{v}$

$\mathsf{t}, \mathsf{t}_1, \mathsf{t}_2 \in \mathbb{R}$; $\mathsf{v}, \mathsf{v}_1, \mathsf{v}_2 \in \mathbb{R}$; $\sim \in \{<, >, =, <>, <=, >=\}$;
s is a signal or a mathematical expression over the signals S defined in property ϕ.

Figure 3. SB-TemPsy-DSL syntax.

3 Background: SB-TemPsy-DSL

SB-TemPsy-DSL [1] is a domain-specific language for expressing requirements that concern signal-based temporal properties. The syntax of SB-TemPsy-DSL is shown in Figure 3; optional items are enclosed in square brackets; the

$\lambda \models \phi_1$ **or** ϕ_2 *iff* $(\lambda \models \phi_1) \vee (\lambda \models \phi_2)$; $\quad \lambda \models\ \delta_1$ **and** δ_2 *iff* $(\lambda \models \delta_1) \wedge (\lambda \models \delta_2)$; $\quad \lambda \models$ **not** sc *iff* $(\lambda \not\models \mathsf{sc})$

$\lambda \models$ **before** $\mathsf{t}\ \mathsf{p}$ *iff* $t_i < \mathsf{t} \leq t_e \wedge \lambda, [t_i, \mathsf{t}] \models \mathsf{p}$; $\quad \lambda \models$ **between** n **and** $\mathsf{m}\ \mathsf{p}$ *iff* $t_i \leq \mathsf{n} < \mathsf{m} \leq t_e \wedge \lambda, [\mathsf{n}, \mathsf{m}] \models \mathsf{p}$
$\lambda \models$ **globally** p *iff* $\lambda, [t_i, t_e] \models \mathsf{p}$; $\quad \lambda \models$ **at** $\mathsf{t}\ \mathsf{p}$ *iff* $t_i \leq \mathsf{t} \leq t_e \wedge \lambda, [\mathsf{t}, \mathsf{t}] \models \mathsf{p}$; $\quad \lambda \models$ **after** $\mathsf{t}\ \mathsf{p}$ *iff* $t_i \leq \mathsf{t} < t_e \wedge \lambda, [\mathsf{t}, t_e] \models \mathsf{p}$
$\lambda \models$ **before** $\mathsf{p}_1\ \mathsf{p}$ *iff* $\forall t_1, t_2, ((t_i < t_1 < t_2 \leq t_e \wedge \lambda, [t_1, t_2] \models \mathsf{p}_1) \Rightarrow \exists t_3, t_4, (t_i \leq t_3 < t_4 < t_1 \wedge \lambda, [t_3, t_4] \models \mathsf{p}))$
$\lambda \models$ **after** $\mathsf{p}_1\ \mathsf{p}$ *iff* $\forall t_1, t_2, ((t_i \leq t_1 < t_2 < t_e \wedge \lambda, [t_1, t_2] \models \mathsf{p}_1) \Rightarrow \exists t_3, t_4, (t_2 < t_3 < t_4 \leq t_e \wedge \lambda, [t_3, t_4] \models \mathsf{p}))$
$\lambda \models$ **between** p_1 **and** $\mathsf{p}_2\ \mathsf{p}$ *iff* $\forall t_1, t_2, t_3, t_4, ((t_i \leq t_1 < t_2 < t_3 < t_4 \leq t_e \wedge \lambda, [t_1, t_2] \models \mathsf{p}_1 \wedge \lambda, [t_3, t_4] \models \mathsf{p}_2) \Rightarrow \lambda, [t_2, t_3] \models \mathsf{p})$

$\lambda, [t_l, t_u] \models$ **assert** c *iff* $\forall t \in [t_l, t_u], (\lambda, t \models \mathsf{c})$. For every time instant t within $[t_l, t_u]$, condition c holds.
$\lambda, [t_l, t_u] \models \mathsf{s}$ **becomes** $\sim \mathsf{v}$ *iff* $\exists t \in (t_l, t_u], (\mathsf{s}(t) \sim \mathsf{v} \wedge \forall t_1 \in [t_l, t), (\mathsf{s}(t_1) \not\sim \mathsf{v}))$. Formula $\mathsf{s}(t) \sim \mathsf{v}$ is *true* for some t, and for any time instant t_1 before t, $\mathsf{s}(t) \sim \mathsf{v}$ is *false*.
$\lambda, [t_l, t_u] \models$ **exists spike in** s [**with** [**width** $\sim_1 \mathsf{v}_1]_\beta$[**amplitude** $\sim_2 \mathsf{v}_2]_\gamma]_\alpha$ *iff* $\exists t_1, t_2, t_3, t_4, t_5 \in [t_l, t_u], (t_1 < t_2 < t_2 < t_3 < t_4 < t_5 \wedge uni\_m\_max(s, t_2, [t_1, t_3]) \wedge uni\_sm\_min(s, t_3, [t_2, t_4]) \wedge uni\_m\_max(s, t_4, [t_3, t_5])[[\wedge (t_3 - t_1) \sim_1 \mathsf{v}_1]_\beta [\wedge \max((\mathsf{s}(t_2) - \mathsf{s}(t_3)), (\mathsf{s}(t_4) - \mathsf{s}(t_3))) \sim_2 \mathsf{v}_2]_\gamma]_\alpha)$. Signal s has a strict maximum within two (non strict) minima. The values v_1 and v_2 constrain the width and the amplitude of the spike.*
$\lambda, [t_l, t_u] \models$ **exist oscillation in** s [**with** [**period** $\sim_1 \mathsf{v}_1]_\zeta$[**p2pAmp** $\sim_2 \mathsf{v}_2]_\epsilon]_\delta$ *iff* $\exists t_1, t_2, t_3, t_4, t_5 \in [t_l, t_u], (t_1 < t_2 < t_2 < t_3 < t_4 < t_5 \wedge uni\_sm\_max(s, t_2, [t_1, t_3]) \wedge uni\_sm\_min(s, t_3, [t_2, t_4]) \wedge uni\_sm\_max(s, t_4, [t_3, t_5])[[\wedge (t_4 - t_2) \sim_1 \mathsf{v}_1]_\zeta [\wedge (\mathsf{s}(t_2) - \mathsf{s}(t_3)) \sim_2 \mathsf{v}_2 \wedge (\mathsf{s}(t_4) - \mathsf{s}(t_3)) \sim_2 \mathsf{v}_2]_\epsilon]_\delta)$. Signal s shows a strict maximum within two strict minima. The values v_1 and v_2 constrain the period and the amplitude of the oscillation.*
$\lambda, [t_l, t_u] \models \mathsf{s}$ **rises** [**monotonically**]$_\alpha$ **reaching** v *iff* $\exists t \in (t_l, t_u], (\mathsf{s}(t) \geq \mathsf{v} \wedge \forall t_1 \in [t_l, t), (\mathsf{s}(t_1) < \mathsf{v})[\wedge\, monot(\mathsf{s}, t_l, t)]_\alpha)$. There exists a time instant t where $\mathsf{s}(t) \geq \mathsf{v}$, and for any time instant t_1 before t, $\mathsf{s}(t_1) < \mathsf{v}$. The character α labels the formula indicating that the signal shall rise monotonically.
$\lambda, [t_l, t_u] \models \mathsf{s}$ **overshoots** [**monotonically**]$_\alpha$ v_1 **by** v_2 *iff* $\exists t \in (t_l, t_u], (\mathsf{s}(t) \geq \mathsf{v}_1 \wedge \forall t_1 \in [t, t_u], (\mathsf{s}(t_1) \leq \mathsf{v}_1 + \mathsf{v}_2) \wedge \forall t_2 \in [t_l, t)(\mathsf{s}(t_2) < \mathsf{v}_1)[\wedge\, monot(\mathsf{s}, t_l, t)]_\alpha)$. Signal s is initially lower than v_1. It then exceeds v_1 at time instant t by remaining below $\mathsf{v}_1 + \mathsf{v}_2$. The character α labels the formula indicating that the signal shall overshoot monotonically.
$\lambda, [t_l, t_u] \models$ **if** p_1 **then** [**within** $\bowtie\ \mathsf{d}]_\alpha\ \mathsf{p}_2$ *iff* $\forall t_1, t_2 \in [t_l, t_u), ((t_1 < t_2 \wedge \lambda, [t_1, t_2] \models \mathsf{p}_1) \Rightarrow \exists t_3, t_4 \in [t_2, t_u], (t_3 < t_4 \wedge \lambda, [t_3, t_4] \models \mathsf{p}_2[\wedge (t_3 - t_2)[\![\bowtie]\!]\mathsf{d}]_\alpha))$ where $[\![\bowtie]\!]$ is such that [[**exactly**]] ≡ '=', [[**at most**]] ≡ '<=', [[**at least**]] ≡ '>='. If pattern p_1 holds in an interval $[t_1, t_2]$, then pattern p_2 holds in a subsequent interval $[t_3, t_4]$.

$\lambda, t \models \mathsf{c}_1$ **and** c_2 *iff* $(\lambda, t \models \mathsf{c}_1) \wedge (\lambda, t \models \mathsf{c}_2)$; $\quad \lambda, t \models \mathsf{c}_1$ **or** c_2 *iff* $(\lambda, t \models \mathsf{c}_1) \vee (\lambda, t \models \mathsf{c}_2)$; $\quad \lambda, t \models \mathsf{s} \sim \mathsf{v}$ *iff* $s(t) \sim \mathsf{v}$

$\mathsf{t}, \mathsf{t}_1, \mathsf{t}_2 \in \mathbb{R}$; $\mathsf{v}, \mathsf{v}_1, \mathsf{v}_2 \in \mathbb{R}$; $\sim \in \{<, >, =, \neq, \leq, \geq\}$; s is a signal in S or a mathematical expression over the signals in S.

$monot(\mathsf{s}, t_1, t_2) ::= \forall t_3 \in [t_1, t_2), \forall t_4 \in (t_3, 2], (\mathsf{s}(t_3) < \mathsf{s}(t_4))$.

$uni\_m\_max(\mathsf{s}, t, [t_a, t_b]) ::= \mathsf{s}(t) = x$ and $\forall t_1 \in [t_a, t_b], \mathsf{s}(t_1) < x$ and $\forall t_1, t_2 \in [t_a, t],$ if $t_1 < t_2$ then $\mathsf{s}(t_1) \leq \mathsf{s}(t_2)$ and $\forall t_1, t_2 \in [t_a, t],$ if $t_1 < t_2$ then $\mathsf{s}(t_1) \geq \mathsf{s}(t_2)$

$uni\_sm\_max(\mathsf{s}, t, [t_a, t_b]) ::= \mathsf{s}(t) = x$ and $\forall t_1 \in [t_a, t_b], \mathsf{s}(t_1) < x$ and $\forall t_1, t_2 \in [t_a, t],$ if $t_1 < t_2$ then $\mathsf{s}(t_1) < \mathsf{s}(t_2)$ and $\forall t_1, t_2 \in [t_a, t],$ if $t_1 < t_2$ then $\mathsf{s}(t_1) > \mathsf{s}(t_2)$

* We present the case where a (strict) minimum is followed by a strict maximum followed by a (strict) minimum. The dual case can be derived from our formulation. Similarly, we present the predicates that characterize a local (strict) maximum (uni_m_max and uni_sm_max). Their dual case, i.e., the predicates that characterize a local (strict) minimum (uni_m_min and uni_sm_min) can be derived from the above formulations.

Figure 4. SB-TemPsy-DSL formal semantics (based on [1])

symbol '|' separates alternatives.[1] A *property* is a disjunction of clauses. A *clause* is a conjunction of atoms. An *atom* is defined in terms of a *scope* (non-terminal sc) or the negation operator (**not**) applied to constructs of type scope (sc). A scope operator constrains a *pattern* (non-terminal p) to hold within a given time interval. There are two types of scope operators: absolute scopes and event scopes. Absolute scopes are delimited by absolute time instants (e.g., **before** $\mathsf{t}\ \mathsf{p}$). Event scopes are delimited by other patterns (e.g., **before** $\mathsf{p}_1\ \mathsf{p}$). A pattern (e.g., **exists spike in** s [...]) specifies a constraint on the behavior of one or more signals. A *condition*, which is used within the **assert** c pattern, is a comparison ($\mathsf{s} \sim \mathsf{v}$) between the value of a signal s and the value v, or a combination of two conditions with the **and** and **or** logical operators.

SB-TemPsy-DSL supports the following patterns:

- **assert** indicates an event-based data assertion. It specifies a constraint on the value of a signal. A requirement with the **assert** construct is as follows:
  $R2$: The beta angle shall vary between 90° and −90°. The corresponding SB-TemPsy-DSL specification of the pattern is: $\mathsf{p}_2 \equiv$ **assert** β **<=** 90 **and** β **>=** $-\,90$
- **becomes** represents a state-based data assertion. It specifies a state of the signal, within a specific time interval, that satisfies a condition, which was not satisfied before that interval. A requirement with the **becomes** construct is as follows:
  $R3$: the value of signal $RWS\_command$ shall become greater than 0. The corresponding SB-TemPsy-DSL specification of the pattern is:
  $\mathsf{p}_3 \equiv RWS\_command$ **becomes** > 0
- **rises** indicates a constraint on the transient behavior of a signal, while it reaches, possibly monotonically, a target value. Its dual behavior is called **falls**.
  A requirement with the **rises** construct is as follows:
  $R4$: The X_cur signal of the sun sensor shall rise monotonically reaching the value of $3650\,\mu$A. The corresponding SB-TemPsy-DSL specification of the pattern is:
  $\mathsf{p}_4 \equiv X\_cur$ **rises monotonically reaching** 3650
- **overshoots** specifies a maximum value (i.e., above the target value) that a signal can reach when overshooting (i.e., when it exceeds the target value). The pattern can possibly be defined with a monotonicity constraint that requires a monotonic increase of the signal prior to reaching its target value. The dual behavior of this pattern is called *undershoot* and is expressed with the keyword **undershoots**.

[1] The grammar of SB-TemPsy-DSL considered in this paper is slightly different from the original one [1]. Any SB-TemPsy-DSL property can be rewritten following this grammar by using standard rewriting rules [22].

A requirement with the **overshoots** construct is as follows:
$R5$: The *X_cur* signal of the sun sensor shall monotonically overshoot the value of $3650\,\mu\mathrm{A}$ by at most $50\,\mu\mathrm{A}$. The corresponding SB-TemPsy-DSL specification of the pattern is:
$\mathsf{p}_5 \equiv X\_cur$ **overshoots monotonically** 3650 **by** 50.

- **spike** specifies a large increase (or decrease) of the value of a signal. A spike is characterized by three extrema (one strict maximum surrounded by two local minima if it represents an increase of the signal, or one strict minimum surrounded by two local maxima if it represents a decrease of the signal). A requirement with the **spike** construct is as follows:
  $R6$: The *beta_angle* signal shall show a spike with an amplitude less than 90°. The corresponding SB-TemPsy-DSL specification of the pattern is:
  $\mathsf{p}_6 \equiv$ **exists spike in** $beta\_angle$
  with amplitude < 90.
- **oscillation** specifies a repeated variation, over time, of the signal value. During an oscillatory behavior, the signal value swings from one extremum to the adjacent extremum of the same type (i.e., maximum or minimum) by traversing an extremum of the other type. A requirement with the **oscillation** construct is as follows:
  $R7$: The velocity of the satellite along the *X_axis* signal shall oscillate with a maximum amplitude of $8000\,\mathrm{km}$ per hour and a maximum period of $180\,\mathrm{min}$. The corresponding SB-TemPsy-DSL specification of the pattern is:
  $\mathsf{p}_7 \equiv$ **exist oscillation in** X_axis
  with p2pAmp $<= 8000$
  with period $<=\ 180$.
- **if-then** represents a constraint on a response behavior of one or two signals, where a pattern (i.e., effect pattern) shall hold some time after a trigger pattern (i.e., cause pattern) has held in the past. A requirement with the **if-then** construct is as follows:
  $R8$: If the value of signal *not_Eclipse* is equal to 0, then the value of signal *sun_currents* should eventually be equal to 0. The corresponding SB-TemPsy-DSL specification is:
  $\mathsf{p}_8 \equiv$ **if** $not\_Eclipse\ = 0$ **then** $sun\_currents\ = 0$

The syntax of SB-TemPsy-DSL enables engineers to define the property ϕ_1 expressing requirement R1 (see Section 2) as: $\phi_1 \equiv$ **globally exists spike in** β **with width** <0.5 **amplitude** < 90. Note that this property is made by a single atom (represented by the **globally** scope construct, which is applied to a pattern p).

The semantics of each construct η of SB-TemPsy-DSL is shown in Figure 4, which is divided into four parts. The first part contains the semantics of properties, clauses, and atoms. The other three parts address the semantics of scopes, patterns, and conditions. The semantics of a construct η is the formula $\zeta(\eta)$ (in first-order logic) written on the right side of the *iff* (if and only if) sign.

Recall from section 2 that a trace λ is a sequence of records that describe how the values of one or more signals in the set $S = \{s_1, s_2, \dots, s_n\}$ change over time. More precisely, each record contains a timestamp, identifying the time at which the record was collected, and the values assumed by some variables, each recording the values of one of the monitored signals in S at that time. For properties, the semantics specifies the conditions that make a property ϕ satisfied by the trace λ, i.e., $\lambda \models \phi$. For example, the semantics of ϕ_1 **or** ϕ_2 requires at least one of them to hold on trace λ. For scopes, the semantics specifies the conditions that make a scope sc satisfied on the trace λ, i.e., $\lambda \models \mathsf{sc}$. For example, the semantics of the **globally** scope indicates that a pattern p scoped by the **globally** operator holds on the trace λ if the pattern p holds on the interval of the trace λ delimited by the timestamps t_i and t_e. The timestamps t_i and t_e indicate the initial and the last timestamps of the trace. For patterns, the semantics specifies the conditions that make a pattern p satisfied on the interval of the trace λ delimited by the timestamps t_l and t_u defined by a given scope, i.e., $\lambda, [t_l, t_u] \models \mathsf{p}$. Figure 4 also includes an informal description of the pattern semantics, after the formal definition. For example, the semantics of pattern "**exists spike in** s **with width** $\sim_1 \mathsf{v}_1$ **amplitude** $\sim_2 \mathsf{v}_2$" specifies that signal s shows a spike behavior with a width satisfying the constraint "$\sim_1 \mathsf{v}_1$" and an amplitude satisfying the constraint "$\sim_2 \mathsf{v}_2$". A spike informally denotes a temporary (large) increase (or decrease) of the value of a signal. It occurs when the signal has a strict maximum surrounded by two minima (or a strict minimum surrounded by two maxima). This behavior can be subjected to additional constraints on the width (i.e., the difference between the time instants at which the two minima — or the two maxima occur) and on the amplitude (i.e., the difference between the maximum and minimum values of the signal). Finally, the semantics of conditions specifies how to satisfy a condition c for a trace λ at time instant t, i.e., $\lambda, t \models \mathsf{c}$. For example, the semantics of c_1 **and** c_2 requires both c_1 and c_2 to hold on the trace λ at timestamp t.

SB-TemPsy-DSL is supported by SB-TemPsy-Check [1], an automated, model-driven trace-checking tool that verifies whether a property is satisfied or violated by a given trace. SB-TemPsy-Check yields a Boolean verdict: *true* if the property is satisfied, *false* otherwise. For example, when property ϕ_1 is checked on the trace shown in Figure 1, SB-TemPsy-Check returns the *false* verdict. However, SB-TemPsy-Check does not provide any additional information to help engineers understand the cause of the violation. Our trace diagnostic approach aims to solve this problem.

4 Pattern-based trace diagnostic

This section describes *TD-SB-TemPsy*, our trace-diagnostic approach. At the core of the approach, there is the computation of *violation causes* and *diagnoses*. A violation cause characterizes one of the possible behaviors of the system that may lead to the property violation. An example of violation cause for property ϕ_1 and the trace in Figure 2 is that all the spikes have an amplitude greater than or equal to 90°. A diagnosis provides additional information to explain the violation cause. For example, a diagnosis for the previous violation cause is the amplitude $A_1 = 150°$ and the time interval $[0\,\mathrm{s}, 1.8\,\mathrm{s}]$ of spike *spike1*, that is the closest (among those contained in the trace) to satisfy the amplitude constraint of property ϕ_1.

Algorithm 1 *TD-SB-TemPsy*

Inputs. λ: trace
ϕ: violated property
Outputs. *diags*: set of diagnoses instances

```
1: function TD-SB-TEMPSY($\lambda$, $\phi$)
2:     diags={};
3:     PropertyAtoms=GETATOMS($\phi$);
4:     for $\alpha$ in PropertyAtoms do
5:         if CHECKATOMONTRACE($\lambda$,$\alpha$)==false then
6:             diags.add(TD-ATOM($\lambda$,$\alpha$))
7:     return diags;
```

Algorithm 2 *TD-SB-TemPsy* - Atoms

Inputs. λ: trace
α: violated atom
Outputs. *diag*: diagnosis for α (if available)

```
1: function TD-ATOM($\lambda$,$\alpha$)
2:     vcs=GETVIOLATIONCAUSES($\alpha$);
3:     for i=0; i<vcs.size(); i++ do
4:         if CHECKVIOLATIONCAUSE($\lambda$,vcs[i])==true then
5:             return GETDIAGNOSIS($\lambda$,vcs[i]);
6:     return null;
```

Algorithm 1 shows the main steps of *TD-SB-TemPsy*. The inputs of *TD-SB-TemPsy* are a trace λ and a property ϕ violated by λ. Trace λ is a set of consecutive records that contain the values of the variables at different time instants, such as the trace depicted in Figure 1. Property ϕ is a specification of a requirement in SB-TemPsy-DSL defined according to the grammar presented in Figure 3.

The algorithm relies on the following intuition. Based on the SB-TemPsy-DSL grammar shown in Figure 3, property ϕ is specified as a disjunction of clauses. Since the property is violated, the disjunction evaluates to false; this means that all its clauses must be violated. To be violated, each clause must contain one or more violated atoms (since a clause is a conjunction of atoms). Therefore, to explain the violation of property ϕ, we return the diagnoses (if available[2]) for all the violated atoms of ϕ. Each diagnosis explains why the corresponding atom is violated.

Algorithm 1 works as follows. After initializing a set of diagnoses instances to be returned (line 2), it extracts all the atoms from property ϕ by analyzing its abstract syntax tree (line 3). Then, it iteratively analyzes each atom (line 4). It first checks, by calling a trace checker for SB-TemPsy-DSL like SB-TemPsy-Check [1], if the atom is violated by the trace (line 5). If it is the case, the algorithm computes (through algorithm TD-ATOM described below) the diagnosis (if available) that explains why the atom is violated (line 6). Finally, the algorithm returns the set of the computed diagnoses instances (line 7). *TD-SB-TemPsy* returns a diagnosis if the set of the computed diagnoses instances is not empty.

Computing the diagnosis. Algorithm 2 describes how to compute the diagnosis for an atom of an SB-TemPsy-DSL property. The inputs of TD-ATOM are a trace λ and an atom α violated by λ.

To compute the diagnosis for atom α, Algorithm 2 extracts the *violation causes* associated with the atom (line 2) by calling the auxiliary function GETVIOLATIONCAUSES, which relies on a predefined mapping associating each type of atom of SB-TemPsy-DSL with one or more violation causes (see section 6). A violation cause encodes a behavior that may lead to the violation of an atom. For example, a violation cause for the atom of property ϕ_1 defined in Section 2 is the following:

c_spike$_1$: *all the spikes in signal β violate the amplitude constraint.*

If the behavior captured by this violation cause holds, the atom of formula ϕ_1 is violated. The violation of an atom can be caused by a violation of the scope used in the atom (or its negation), a violation of the pattern constrained by the scope, or both. Function GETVIOLATIONCAUSES returns a list of violation causes, sorted such that violation causes of the scope precede the violation causes of the pattern[3]. Then, the algorithm loops through this list of violation causes (line 3). The loop body includes a check that determines whether the violation cause holds on the trace (line 4); this is achieved through the call of the auxiliary function CHECKVIOLATIONCAUSE. If the violation cause holds, the algorithm stops, returning the corresponding diagnosis (line 5) using the auxiliary function GETDIAGNOSIS, which relies on a predefined mapping of diagnoses for each type of violation cause (see section 6). A diagnosis is relevant information that enables engineers to understand *why a violation cause holds on a trace*. For example, the diagnosis for the violation cause **c_spike**$_1$ is the following

d_spike$_1$: *the amplitude* a *and the time interval* $[\mathrm{t}_1, \mathrm{t}_2]$ *of the spike that is the closest to satisfy the amplitude constraint.*

The amplitude value of the spike that is the closest to satisfy the amplitude constraint enables engineers to determine how close is the atom to be satisfied. The time interval $[\mathrm{t}_1, \mathrm{t}_2]$ enables engineers to isolate the portion of the trace containing that spike and to inspect the values assumed by the variables within the records included in this portion of the trace. The amplitude $A_1 = 150°$ and the time interval $[0, 1.8]$ of *spike1* in Figure 2 is an instance of the **d_spike**$_1$ diagnosis for our case study. Diagnoses like **d_spike**$_1$ help engineers understand why an atom is violated by a trace. If all the violation causes are checked and none of them led to the computation of diagnoses, a null value is returned (line 6).

In the following, we present our methodology to define violation causes and diagnoses.

5 Methodology for Defining Violation Causes and Diagnoses

This section describes our methodology to define violation causes and diagnoses by using SB-TemPsy-DSL as an example. Our methodology considers each construct η used to define an atom α of SB-TemPsy-DSL and follows three steps: *behavior analysis*, *definition of violation causes*, and *definition of diagnoses*.

[2] As we will discuss later on, it is possible for a violated atom not to have any diagnosis.

[3] The priority of the different violation causes is application-specific. Our current implementation is based on the feedback received by the engineers of our industrial partners

5.1 Behavior Analysis

It identifies traces capturing relevant behaviors that violate the semantics of construct η as follows.

1) It considers an instance of construct η obtained by selecting some values for its parameters. This instance is a concrete example utilized to identify the relevant behaviors that violate η. For example, for the **spike** construct of SB-TemPsy-DSL (see Figure 3), we considered the instance "**exists spike in** β **with width** < 0.5 **amplitude** < 90 ", which sets the parameters s, $\sim_1$, v_1, $\sim_2$, and v_2 to the values β, "<", 0.5, "<", and 90 respectively.
2) It considers the logical formula $\zeta(\eta)$ describing the semantics of the construct η. For example, for the **spike** construct the logical formula describing its semantics is reported in Figure 4 (on the right side of the *iff* operator).
3) It identifies traces capturing relevant behaviors that violate the instance of construct η (i.e., that make formula $\zeta(\eta)$ evaluate to *false*).
For example, Figure 5 shows a trace with four signals (β_1, β_2, β_3, and β_4) that violate the instance we considered for the **spike** construct; for instance, signal β_4 does not contain a strict maximum.

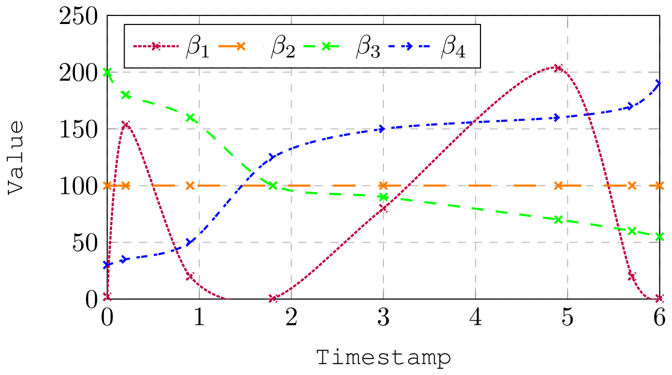


Figure 5. A trace with signals violating the expression "**exists spike in** β **with width** < 0.5 **amplitude** < 90".

5.2 Definition of Violation Causes

It characterizes each of the traces identified by the behavior analysis step through a violation cause. Each violation cause vc is defined by writing a logical formula $\zeta(vc)$ that specifies its semantics; the formula $\zeta(vc)$ is true when trace λ satisfies violation cause vc.

For example, the shape of signal β_3 can be defined, with respect to a trace λ delimited by timestamps t_i and t_e, through the logical formula:

$$\texttt{c\_spike}_4 \equiv \forall t_1 \in [t_l, t_u), (\forall t_2 \in (t_1, t_u], (\mathsf{s}(t_1) \geq \mathsf{s}(t_2))).$$

This formula characterizes the behavior for which a signal s decreases. More precisely, **c_spike**$_4$ holds on a trace λ if, for any timestamp t_1 within t_l and t_u, the value $\mathsf{s}(t_1)$ of signal s at timestamp t_1 is greater than or equal to the value $\mathsf{s}(t_2)$ of signal s for any timestamp t_2 that follows t_1.

Violation causes for SB-TemPsy-DSL are illustrated in section 6 (and summarized in Figure 7). The formula $\zeta(vc)$ characterizing the semantics of each violation cause is reported on the right side of the *iff* operator.

Since violation causes should encode root causes leading to property violations, a violation cause vc for a construct η should satisfy the following relation:

$$\textbf{if } \zeta(vc) = \texttt{true}, \textbf{ then } \zeta(\eta) = \texttt{false}$$

Intuitively, if violation cause vc holds on trace λ, i.e., formula $\zeta(vc)$ is true, the trace λ should violate construct η, i.e., formula $\zeta(\eta)$ should be false. To check the satisfaction of this relation, we automatically verified that the formula

$$\Psi \equiv \zeta(vc) \Rightarrow \neg\zeta(\eta)$$

holds. Formula Ψ holds if, whenever the violation cause holds ($\zeta(vc)$ is true), the construct η does not hold ($\zeta(\eta)$ is false). To check if formula Ψ holds, we verified whether the formula $\neg\Psi$ is unsatisfiable. If the formula $\neg\Psi$ is unsatisfiable, Ψ always holds. We used Microsoft Z3 [23] — an industry-strength tool — to check if $\neg\Psi$ is unsatisfiable. For example, Z3 confirmed that the formula $\neg\Psi$ obtained by considering the **c_spike**$_4$ violation cause and the **spike** construct is unsatisfiable. Therefore, whenever violation cause **c_spike**$_4$ holds, the **spike** construct is violated.

5.3 Definition of diagnoses

It defines a diagnosis for each of the violation causes. To define each diagnosis, we (i) analyzed the semantics of the corresponding violation cause, and (ii) identified minimum relevant information that enables engineers to understand why a violation cause holds on a trace.

For example, the diagnosis for violation cause **c_spike**$_4$, denoted by **d_spike**$_4$, includes the lowest and the highest values of signal s within the trace. These values allow engineers to understand the range of values of signal s while it exhibits a decreasing behavior. For example, for signal β_3 in Figure 5, the diagnosis for violation cause **c_spike**$_4$ is $\langle\langle 0, 200\rangle, \langle 6, 55\rangle\rangle$ showing that signal β_3 reaches its maximum value (200) at timestamp 0 and its minimum value (55) at timestamp 6. In this particular case, when the signal does not show any spike, this is some information required for the engineers to understand why this particular violation was caused.

The diagnoses associated with the violation causes shown in Figure 7 are illustrated in section 6 (and summarized in Figures 8–9).

5.4 Properties of the methodology

Our methodology provides formal guarantees of the soundness of the proposed violation causes: if a violation cause holds on a trace, the corresponding property is violated.

We remark that the methodology relies mostly on manual steps (except for proving, using a solver like Z3, that the proposed violation causes can lead to unsatisfiable properties). Moreover, the methodology cannot guarantee the completeness of the set of violation causes resulting from step "Definition of violation causes".

Though we applied our methodology to define violation causes for properties expressed using SB-TemPsy-DSL, our methodology is *language-agnostic*, in the sense that *it can* *be applied to other pattern-based languages* like TemPsy [17], FRETISH [18] and, more in general, languages based on specification pattern catalogues (such as those for robotic missions [7]). More specifically, this claim is supported by the fact that, in the definition of the methodology, we only refer to generic syntactic constructs of the specification language and generic logical formulae capturing their semantics (which we assume to be available in formal syntax and semantics definitions), as well as generic logical formulae of violation causes (which have to be defined from scratch). The restriction to pattern-based languages is due to the fact that we assume the existence of some pattern-based structure in the syntax of the specification language. We also remark that the methodology would not work for specification languages in which the Boolean and temporal operators are less constrained (e.g., STL).

For the sake of illustration, in the rest of this subsection, we show how to apply the methodology to one construct supported by FRETISH.

Behavior Analysis

Let us consider the FRETISH scope construct **`only in mode`** [24]; it informally indicates that a requirement shall only hold within the time interval delimited by the scope boundaries (also called endpoints). More specifically, it restricts the satisfaction of that requirement to only the time interval delimited by the scope boundaries. In other words, this implies that the requirement shall be violated within the remaining time interval(s) outside the interval determined by the scope boundaries. Let us consider the following requirement

RF: *"Only in mode M, signal s shall be less than or equal to 3"*.

The corresponding FRETISH specification is the following:

$$\phi_f \equiv \quad \texttt{only in } M \texttt{ mode } s \texttt{ shall satisfy } s < 3$$

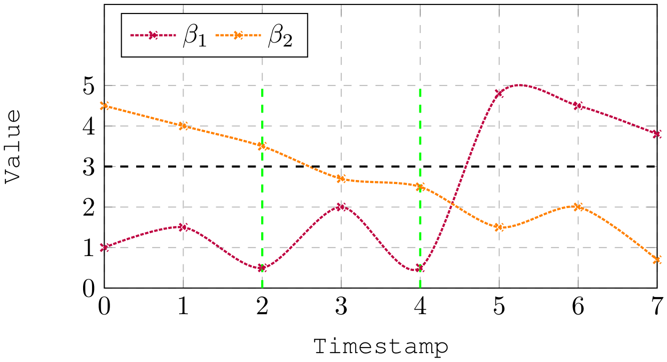


Figure 6. A trace with signals violating the expression "**`only in`** M **`mode`** s **`shall satisfy`** s < 3".

The logical formula describing the formal semantics of a property with the **`only in mode`** construct is defined as follows:
$\lambda \models \texttt{only in}\ \mathsf{M}\ \texttt{mode}\ \mathsf{s}(t) \sim \mathsf{v}$ *iff* $\forall t_1 \in [\mathit{ftp}, \mathit{fim}), \mathsf{s}(t) \not\sim \mathsf{v} \wedge \forall t_2 \in (\mathit{lim}, \mathit{ltp}], \mathsf{s}(t) \not\sim \mathsf{v}$. Informally, given a trace length delimited by the *ftp* and *ltp* time points where *ftp* < *ltp*, and a time interval $[\mathit{fim}, \mathit{lim}]$ in which mode M holds, where $\mathit{fim} \geq \mathit{ftp}$ and $\mathit{lim} \leq \mathit{ltp}$, the definition of the semantics of the **`only in mode`** construct indicates that the property condition ($s \sim \mathsf{v}$) shall not be satisfied outside of the scope interval $[\mathit{fim}, \mathit{lim}]$.

We identify two possible relevant behaviors that violate an instance of the **`only in mode`** construct (i.e., that make formula ϕ_f evaluate to *false*).

As depicted in Figure 6, let us consider a trace defined within the time horizon delimited by the first time point *ftp* and the last time point *ltp* (0 and 7 in the figure, respectively). M holds within the time interval $[2, 4]$ (delimited by the dashed green lines in the figure) where timestamp 2 is referred to as *fim* (first in mode) and timestamp 4 represents the last timestamp in mode M (*lim*), such that $\mathit{ftp} \leq \mathit{fim} < \mathit{lim} \leq \mathit{ltp}$. For property ϕ_f *to be violated*, $s < 3$ shall be satisfied outside the interval $[\mathit{fim}, \mathit{lim}]$; this means that $s < 3$ is satisfied either within the time interval $[\mathit{ftp}, \mathit{fim})$ or within the time interval $(\mathit{lim}, \mathit{ltp}])$. For instance, signal β_1 in the figure satisfies the condition, since its value is less than 3 (i.e., ranging between 1 and 1.5) in the time interval $[0, 1]$ before mode M holds (i.e., in the time interval $[2, 4]$). Similarly, signal β_2 in the figure satisfies the condition, since its value ranges between 0.7 and 2 (1.5, 2 and 0.7) in the time interval $[5, 7]$, which is an interval in which mode M does not hold.

Definition of Violation Causes

In the following, we define and formalize two violation causes that, when satisfied, lead to the violation of property ϕ_f.

- **`c_only_in_mode`**$_1$: Some time before mode M starts holding, the signal satisfies the property condition. However, the signal does not satisfy the constraint after mode M stops holding. Formally, the violation cause is defined through the following logical formula:
$\lambda \models \texttt{c\_only\_in\_mode}_1$ *iff* $\exists t_1 \in [\mathit{ftp}, \mathit{fim}), \mathsf{s}(t) \sim \mathsf{v} \wedge \forall t_2 \in (\mathit{lim}, \mathit{ltp}], \mathsf{s}(t) \not\sim \mathsf{v}$.
For instance, signal β_1 in the figure takes values ranging between 1 and 1.5 in the time interval $[0, 1]$ before mode M started holding. However, β_1 violates the constraint (i.e., $\beta_1 \geq 3$) by taking values ranging between 3.8 and 4.8 (4.8, 4.5 and 3.8) within the time interval $[5, 7]$, after mode M stopped holding.
- **`c_only_in_mode`**$_2$: Some time after mode M stops holding, the signal satisfies the property condition. However, the signal does not satisfy the constraint before mode M starts holding. Formally, the violation cause is defined through the following logical formula:
$\lambda \models \texttt{c\_only\_in\_mode}_2$ *iff* $\exists t_2 \in (\mathit{lim}, \mathit{ltp}], \mathsf{s}(t) \sim \mathsf{v} \wedge \forall t_1 \in [\mathit{ftp}, \mathit{fim}), \mathsf{s}(t) \not\sim \mathsf{v}$. For instance, signal β_2 in the figure takes values ranging between 0.7 and 2 (1.5, 2 and 0.7) in the time interval $[5, 7]$. However, β_2 violates the constraint (i.e., $\beta_2 \geq 3$) within the time interval $[0, 1]$ (showing values 4.5 and 4), right before mode M starts holding.

Similar to what we did before with SB-TemPsy-DSL violation causes, we used Z3 to confirm that the formulae $\neg\Psi$ obtained from the two violation causes **`c_only_in_mode`**$_1$ and **`c_only_in_mode`**$_2$ were unsatisfiable.

Definition of diagnoses

We propose the following diagnoses related to the satisfaction of the violation causes `c_only_in_mode`$_1$ and `c_only_in_mode`$_2$:

- `d_only_in_mode`$_1$ includes the first record (timestamp and the corresponding signal value) in which the signal satisfies the property condition, right before mode M starts holding. The choice of this diagnosis is motivated by the fact that we are interested in reporting the root cause of the property violation. More formally, we have:
  `d_only_in_mode`$_1$ $= \langle t, \mathsf{s}(t)\rangle \mid t \in [\mathit{ftp}, \mathit{fim}), \mathsf{s}(t) \sim \mathsf{v} \wedge \forall t_1 \in (\mathit{lim}, \mathit{ltp}], \mathsf{s}(t_1) \not\sim \mathsf{v} \wedge \forall t_2 \in [\mathit{ftp}, t), \mathsf{s}(t_2) \not\sim \mathsf{v}$, where t represents the first timestamp in which the signal satisfies the property condition (i.e., $\mathsf{s}(t) \sim \mathsf{v}$) before mode M starts holding and $\mathsf{s}(t)$ denotes the corresponding signal value.
  For instance, for signal β_1 in the figure, the diagnosis for violation cause `d_only_in_mode`$_1$ is $\langle(0, 1)\rangle$, showing that the signal first satisfies the property condition outside the time interval in which mode M holds (i.e., the signal takes value 1, which is less than 3, at timestamp 0).
- `d_only_in_mode`$_2$ includes the first record in which the signal satisfies the property condition, right after mode M stops holding. Formally, the diagnosis is defined as follows: `d_only_in_mode`$_2$ $= \langle t, \mathsf{s}(t)\rangle \mid t \in (\mathit{lim}, \mathit{ltp}], \mathsf{s}(t) \sim \mathsf{v} \ \wedge\ \forall t_1 \in [\mathit{ftp}, \mathit{fim}), \mathsf{s}(t_1) \not\sim \mathsf{v} \wedge \forall t_2 \in (\mathit{lim}, t), \mathsf{s}(t_2) \not\sim \mathsf{v}$ where t represents the first timestamp in which the signal satisfies the property condition (i.e., $\mathsf{s}(t) \sim \mathsf{v}$) after mode M stops holding and $\mathsf{s}(t)$ denotes the corresponding signal value.
  For instance, for signal β_2 in the figure, the diagnosis for violation cause `d_only_in_mode`$_2$ is $\langle(5, 1.5)\rangle$, showing that the signal first satisfies the property condition (i.e., the signal value is less than 3) at timestamp 5, taking value 1.5.

6 Violation Causes and Diagnoses for SB-TemPsy-DSL

In this section, we describe the *violation causes* and the corresponding *diagnoses* for each *construct* supported by SB-TemPsy-DSL. We first provide a high-level overview through Figure 7 (for *violation causes*) and Figures 8–9 (for *diagnoses*); the remaining subsections discuss in detail the violation causes and diagnoses for each main construct of SB-TemPsy-DSL.

We remark that the violation causes and corresponding diagnoses for SB-TemPsy-DSL have been defined (and validated) together with a group of system and software engineers of our industrial partner, with the goal of maximizing the usefulness of a diagnosis for a certain violation cause. Overall, we spent 20 hours (over three business days) to define the catalogue of violation causes and diagnoses, following the methodology described in section 5.

Figure 7 presents the *violation causes* for the constructs of SB-TemPsy-DSL that can be used in the definition of an *atom* α. It is divided into three parts that respectively contain the violation causes for the SB-TemPsy-DSL atoms, scopes, and patterns. Each violation cause has a name that identifies the construct of SB-TemPsy-DSL the violation cause refers to, and an incremental index that distinguishes violation causes that refer to the same construct; for example, `c_becomes`$_1$, `c_becomes`$_2$, and `c_becomes`$_3$ are the three violation causes that refer to the `becomes` construct of SB-TemPsy-DSL. Each violation cause is parameterized with the *same* parameters as the corresponding construct. For example, the parameters of the `c_becomes`$_1$ violation cause ($\sim$ and v) are the same as those of the `becomes` construct in Figure 3. For conciseness, in Figure 7, we omit the parameters of the violation causes.

The semantics of each violation cause is the (first-order logic) formula $\zeta(vc)$ on the right side of the *iff* operator; it is followed by an informal description of the semantics in English. The semantics of the violation causes specifies the conditions that make the violation causes satisfied by trace λ. For example, the semantics of the violation cause `c_becomes`$_1$ specifies that, for every timestamp t, the value $\mathsf{s}(t)$ does not satisfy $\mathsf{s}(t) \sim \mathsf{v}$. Note that the parameters of the SB-TemPsy-DSL constructs associated with the violation causes, e.g., the value of v, are used to define the semantics of the corresponding violation cause.

Figures 8–9 present the *diagnoses* for the *violation causes* in Figure 7. Figure 8 contains the diagnoses related to the violation causes for SB-TemPsy-DSL atoms and scopes, while Figure 9 contains the diagnoses related to violation causes for patterns.

The name of the diagnosis is obtained by replacing the string "`c_`" with "`d_`" from the name of the corresponding violation cause. For example, diagnosis `d_becomes`$_1$ refers to violation cause `c_becomes`$_1$.

The formal definition of the diagnosis is reported on the right side of the symbol "=". For example, the definition of the diagnosis `d_becomes`$_1$ is the tuple containing the maximum and the minimum values (as well as their timestamps) of signal[4] s. Violation causes sharing the same diagnosis are separated by the symbol "/". For example, `d_a_at`$_1$/`d_a_bef`$_1$/ `d_a_aft`$_1$ is the diagnosis associated with violation causes `c_a_at`$_1$, `c_a_bef`$_1$, and `c_a_aft`$_1$. The *informal definition* provides a high-level description of the diagnosis.

Implementation

We implemented *TD-SB-TemPsy* as an OCL [25] plugin for SB-TemPsy-Check [1]. The plugin contains the definitions of OCL constraints that encode the violation causes (see Figure 7) as well as OCL functions that compute the diagnoses (see Figures 8–9) associated with the violation causes. The full OCL encoding is available at https://github.com/SNTSVV/TD-SB-TemPsy.

6.1 *Patterns*

6.1.1 `assert`*: Event-based Data Assertion*

Violation cause: This pattern is violated if there exists at least one record in the trace that violates the condition used in the assertion. Recall that a record is used

[4] To minimize cluttering, hereafter we omit to indicate that each signal value is associated with a signal name.

$\lambda \models \texttt{c\_not}_1\ \mathsf{sc}$ *iff* $\lambda \models \mathsf{sc}$. The atom sc is satisfied.

$\lambda \models \texttt{c\_a\_at}_1$ *iff* $t < t_i \vee t_e < t$. The value of t is not within the time interval $[t_i, t_e]$.

$\lambda \models \texttt{c\_a\_bef}_1$ *iff* $t \leq t_i \vee t_e < t$. The value of t is not within the time interval $[t_i, t_e]$.

$\lambda \models \texttt{c\_a\_aft}_1$ *iff* $t < t_i \vee t_e \leq t$. The value of t is not within the time interval $[t_i, t_e]$

$\lambda \models \texttt{c\_a\_bet}_1$ *iff* $n < t_i \vee t_e < m \vee m \leq n$. Either the value of n or m is not within $[t_i, t_e]$, or the value of n is not smaller than m.

$\lambda \models \texttt{c\_e\_bef}_1$ *iff* $\exists t_1, t_2, (t_i < t_1 < t_2 \leq t_e \wedge \lambda, [t_1, t_2] \models p_1 \wedge \forall t_3, t_4, (t_i \leq t_3 < t_4 < t_1 \Rightarrow \lambda, [t_3, t_4] \not\models p))$. Pattern p_1 holds within $[t_1, t_2]$. Pattern p is violated before p_1.

$\lambda \models \texttt{c\_e\_aft}_1$ *iff* $\exists t_1, t_2, (t_i \leq t_1 < t_2 < t_e \wedge \lambda, [t_1, t_2] \models p_1 \wedge \forall t_3, t_4, (t_2 < t_3 < t_4 \leq t_e \Rightarrow \lambda, [t_3, t_4] \not\models p))$. Pattern p_1 holds within $[t_1, t_2]$. Pattern p is violated after p_1.

$\lambda \models \texttt{c\_e\_bet}_1$ *iff* $\exists t_1, t_2, t_3, t_4, (t_i \leq t_1 < t_2 < t_3 < t_4 \leq t_e \wedge \lambda, [t_1, t_2] \models p_1 \wedge \lambda, [t_3, t_4] \models p_2 \wedge \lambda, [t_2, t_3] \not\models p)$. Pattern p_1 holds within $[t_1, t_2]$ and pattern p_2 holds within $[t_3, t_4]$, but pattern p does not hold between p_1 and p_2.

$\lambda, [t_l, t_u] \models \texttt{c\_assert}_1$ *iff* $\exists t \in [t_l, t_u], (\lambda, t \not\models \mathsf{c})$. There exists a timestamp t within $[t_l, t_u]$ in which condition c is violated

$\lambda, [t_l, t_u] \models \texttt{c\_becomes}_1$ *iff* $\forall t \in (t_l, t_u], \big(\mathsf{s}(t) \not\sim \mathsf{v}\big)$ .The signal values violate the pattern constraint $\sim \mathsf{v}$ throughout the time interval, delimited by t_l and t_u, over which the pattern is evaluated.

$\lambda, [t_l, t_u] \models \texttt{c\_becomes}_2$ *iff* $\forall t \in (t_l, t_u], \big(\exists t_1 \in [t_l, t), (\mathsf{s}(t_1) \sim \mathsf{v})\big)$. All the signal values observed within the time interval $[t_l, t_u]$ satisfy the pattern constraint $\sim \mathsf{v}$.

$\lambda, [t_l, t_u] \models \texttt{c\_becomes}_3$ *iff* $\exists t \in (t_l, t_u), (\forall t_1 \in [t_l, t), (\mathsf{s}(t_1) \sim \mathsf{v}) \wedge \forall t_2 \in (t, t_u], (\mathsf{s}(t_2) \not\sim \mathsf{v}))$. The signal satisfies the semantics of the pattern instance in which the constraint $\sim \mathsf{v}$ is negated (i.e., $\lambda, [t_l, t_u] \models \mathsf{s}\ \texttt{becomes} \not\sim \mathsf{v}$ holds).

$\lambda, [t_l, t_u] \models \texttt{c\_spike}_1$ *iff* $\forall t_1, t_2, t_3, t_4, t_5 \in [t_l, t_u], ((t_1 < t_2 < t_3 < t_4 < t_5 \wedge \mathit{uni\_m\_min}(\mathsf{s}, t_2, [t_1, t_3]) \wedge \mathit{uni\_sm\_max}(\mathsf{s}, t_3, [t_2, t_4]) \wedge \mathit{uni\_m\_min}(\mathsf{s}, t_4, [t_3, t_5])) \Rightarrow \neg(\mathit{amp}(\mathsf{s}, t_1, t_2, t_3) \sim_2 \mathsf{v}_2))$. All the spike instances violate the amplitude constraint*.

$\lambda, [t_l, t_u] \models \texttt{c\_spike}_2$ *iff* $\forall t_1, t_2, t_3, t_4, t_5 \in [t_l, t_u], ((t_1 < t_2 < t_3 < t_4 < t_5 \wedge \mathit{uni\_m\_min}(\mathsf{s}, t_2, [t_1, t_3]) \wedge \mathit{uni\_sm\_max}(\mathsf{s}, t_3, [t_2, t_4]) \wedge \mathit{uni\_m\_min}(\mathsf{s}, t_4, [t_3, t_5])) \Rightarrow \neg(\mathit{width}(t_2, t_4) \sim_1 \mathsf{v}_1))$. All the spike instances violate the width constraint*.

$\lambda, [t_l, t_u] \models \texttt{c\_spike}_3$ *iff* $\forall t \in [t_l, t_u], (\mathsf{s}(t) = \mathsf{s}(t_l))$. The signal s is constant.

$\lambda, [t_l, t_u] \models \texttt{c\_spike}_4$ *iff* $\forall t_1 \in [t_l, t_u), (\forall t_2 \in (t_1, t_u], (\mathsf{s}(t_1) \geq \mathsf{s}(t_2)))$. The signal s decreases.

$\lambda, [t_l, t_u] \models \texttt{c\_spike}_5$ *iff* $\forall t_1 \in [t_l, t_u), (\forall t_2 \in (t_1, t_u], (\mathsf{s}(t_1) \leq \mathsf{s}(t_2)))$. The signal s increases.

$\lambda, [t_l, t_u] \models \texttt{c\_oscillation}_1$ *iff* $\forall t_1, t_2, t_3, t_4, t_5 \in [t_l, t_u], ((t_1 < t_2 < t_3 < t_4 < t_5 \wedge \mathit{uni\_sm\_min}(\mathsf{s}, t_2, [t_1, t_3]) \wedge \mathit{uni\_sm\_max}(\mathsf{s}, t_3, [t_2, t_4]) \wedge \mathit{uni\_sm\_min}(\mathsf{s}, t_4, [t_3, t_5])) \Rightarrow \neg(\mathit{p2p}(\mathsf{s}, t_2, t_3) \sim_2 \mathsf{v}_2) \wedge \neg(\mathit{p2p}(\mathsf{s}, t_3, t_4) \sim_2 \mathsf{v}_2))$. All the oscillation instances violate the amplitude constraint.

$\lambda, [t_l, t_u] \models \texttt{c\_oscillation}_2$ *iff* $\forall t_1, t_2, t_3, t_4, t_5 \in [t_l, t_u], ((t_1 < t_2 < t_3 < t_4 < t_5 \wedge \mathit{uni\_sm\_min}(\mathsf{s}, t_2, [t_1, t_3]) \wedge \mathit{uni\_sm\_max}(\mathsf{s}, t_3, [t_2, t_4]) \wedge \mathit{uni\_sm\_min}(\mathsf{s}, t_4, [t_3, t_5])) \Rightarrow \neg(\mathit{width}(t_2, t_4) \sim_1 \mathsf{v}_1))$. All the oscillation instances violate the period constraint.

$\lambda, [t_l, t_u] \models \texttt{c\_oscillation}_3$ *iff* $\exists t_1, t_2, t_3 \in [t_l, t_u], (t_1 < t_2 < t_3 \wedge \mathit{ext}(\mathsf{s}, t_2, [t_1, t_3]) \wedge \forall t_4, t_5, t_6 \in [t_l, t_u], ((t_5 \neq t_2 \wedge t_4 < t_5 < t_6) \Rightarrow \neg \mathit{ext}(\mathsf{s}, t_5, [t_4, t_6])))$. The signal s contains only one strict local extremum (minimum or maximum).

$\lambda, [t_l, t_u] \models \texttt{c\_oscillation}_4$ *iff* $\exists t_1, t_2, t_3 \in [t_l, t_u], (t_1 < t_2 < t_3 \wedge \mathit{ext}(\mathsf{s}, t_2, [t_1, t_3]) \wedge \exists t_4, t_5, t_6 \in [t_l, t_u], (t_5 \neq t_2 \wedge t_4 < t_5 < t_6 \wedge \mathit{ext}(\mathsf{s}, t_5, [t_4, t_6]) \wedge \forall t_7, t_8, t_9 \in [t_l, t_u], (t_2 \neq t_8 \neq t_5 \wedge t_7 < t_8 < t_9 \wedge \neg \mathit{ext}(\mathsf{s}, t_8, [t_7, t_9])))))$. The signal s shows only two local extrema.

$\lambda, [t_l, t_u] \models \texttt{c\_oscillation}_5$ *iff* $\forall t \in [t_l, t_u], (\mathsf{s}(t) = \mathsf{s}(t_l))$. The signal s is constant.

$\lambda, [t_l, t_u] \models \texttt{c\_oscillation}_6$ *iff* $\forall t_1 \in [t_l, t_u), (\forall t_2 \in (t_1, t_u], (\mathsf{s}(t_1) \geq \mathsf{s}(t_2)))$. The signal s decreases.

$\lambda, [t_l, t_u] \models \texttt{c\_oscillation}_7$ *iff* $\forall t_1 \in [t_l, t_u), (\forall t_2 \in (t_1, t_u], (\mathsf{s}(t_1) \leq \mathsf{s}(t_2))$. The signal s increases.

$\lambda, [t_l, t_u] \models \texttt{c\_rises}_1$ *iff* $\forall t \in [t_l, t_u], (\mathsf{s}(t) < \mathsf{v})$. The signal value is always below v.

$\lambda, [t_l, t_u] \models \texttt{c\_rises}_2$ *iff* $\forall t \in [t_l, t_u], (\mathsf{s}(t) \geq \mathsf{v})$. The signal value is always greater than or equal to v.

$\lambda, [t_l, t_u] \models \texttt{c\_rises}_3$ *iff* $\exists t \in (t_l, t_u], (\mathsf{s}(t) \geq \mathsf{v} \wedge \forall t_1 \in [t_l, t), (\mathsf{s}(t_1) < v) \wedge \neg(\mathit{mon}(\mathsf{s}, t_l, t))))$. The signal rises at timestamp t, reaching value v. However, it violates the monotonicity constraint defined in the pattern.

$\lambda, [t_l, t_u] \models \texttt{c\_rises}_4$ *iff* $\exists t \in (t_l, t_u), (\forall t_1 \in [t_l, t), (\mathsf{s}(t_1) \geq \mathsf{v}) \wedge \forall t_2 \in [t, t_u], (\mathsf{s}(t_2) < \mathsf{v}))$. The value of the signal is initially above the threshold value v. The signal then drops and remains below that value.

$\lambda, [t_l, t_u] \models \texttt{c\_overshoots}_1$ *iff* $\forall t \in [t_l, t_u], (\mathsf{s}(t) < \mathsf{v}_1)$. The signal s is always below v_1.

$\lambda, [t_l, t_u] \models \texttt{c\_overshoots}_2$ *iff* $\exists t \in [t_l, t_u], (\mathsf{s}(t) > \mathsf{v}_1 + \mathsf{v}_2 \wedge \forall t_1 \in (t, t_u], (\mathsf{s}(t_1) > \mathsf{v}_1 + \mathsf{v}_2))$. The signal s exceeds (and remains above) the value $\mathsf{v}_1 + \mathsf{v}_2$.

$\lambda, [t_l, t_u] \models \texttt{c\_overshoots}_3$ *iff* $\exists t \in (t_l, t_u], (\mathsf{s}(t) \geq \mathsf{v}_1 \wedge \mathsf{s}(t) \leq \mathsf{v}_1 + \mathsf{v}_2 \wedge \forall t_2 \in [t_l, t), (\mathsf{s}(t_2) \leq \mathsf{v}_1) \wedge \forall t_1 \in (t, t_u], (\mathsf{s}(t_1) \leq \mathsf{v}_1 + \mathsf{v}_2) \wedge \neg(\mathit{mon}(\mathsf{s}, t_l, t))))$. The signal overshoots value v_1, without exceeding the maximum threshold set to v_1+ v_2, but it violates the monotonicity constraint.

$\lambda, [t_l, t_u] \models \texttt{c\_overshoots}_4$ *iff* $\exists t \in (t_l, t_u], (\forall t_1 \in [t_l, t], (\mathsf{s}(t_1) \geq \mathsf{v}_1 \wedge \mathsf{s}(t_1) \leq \mathsf{v}_1 + \mathsf{v}_2) \wedge \forall t_2 \in (t, t_u], (\mathsf{s}(t_2) < \mathsf{v}_1)))$. The signal undershoots, going below v_1 after timestamp t, and remains below that value instead of overshooting.

$\lambda, [t_l, t_u] \models \texttt{c\_if-then}_1$ *iff* $\exists t_1, t_2 \in [t_l, t_u), (\lambda, [t_1, t_2] \models \mathsf{p}_1 \wedge (\forall t_3, t_4 \in [t_2, t_u], (\lambda, [t_3, t_4] \not\models \mathsf{p}_2)))$. Pattern p_1 holds within the time interval $[t_1, t_2]$. Pattern p_2 never holds after the satisfaction of pattern p_1, until the end of the time interval, right-bounded by value t_u.

$\lambda, [t_l, t_u] \models \texttt{c\_if-then}_2$ *iff* $\exists t_1, t_2 \in [t_l, t_u), (\lambda, [t_1, t_2] \models \mathsf{p}_1 \wedge \forall t_3, t_4 \in [t_2, t_u], (\lambda, [t_3, t_4] \models \mathsf{p}_2 \Rightarrow \neg((t_3 - t_2)\ [\![\bowtie]\!]\ \mathsf{d})))$ where $[\![\bowtie]\!]$ is such that $[\![\texttt{exactly}]\!] \equiv$ '=', $[\![\texttt{at most}]\!] \equiv$ '<=', $[\![\texttt{at least}]\!] \equiv$ '>='. Pattern p_1 is satisfied within the time interval $[t_1, t_2]$. Any time interval $[t_3, t_4]$ satisfying pattern p_2 violates the time distance constraint on the size of $t_3 - t_2$.

* We present the case where a (strict) minimum is followed by a strict maximum followed by a (strict) minimum. The dual case can be derived from our formulation.

$\mathit{ext}(\mathsf{s}, t_2, [t_1, t_3]) = (\mathit{uni\_sm\_max}(\mathsf{s}, t_2, [t_1, t_3]) \vee \mathit{uni\_sm\_min}(\mathsf{s}, t_2, [t_1, t_3]));\quad \mathit{p2p}(\mathsf{s}, t_1, t_2) = |\mathsf{s}(t_1) - \mathsf{s}(t_2))|;$

$\mathit{amp}(\mathsf{s}, t_1, t_2, t_3) = \max(|\mathsf{s}(t_2) - \mathsf{s}(t_1)|, |\mathsf{s}(t_2) - \mathsf{s}(t_3)|);\quad \mathit{width}(t_1, t_2) = (|t_2 - t_1|)$

Figure 7. Violation causes for the constructs of SB-TemPsy-DSL

`d_not_assert`$=\langle t, \mathsf{s}_1(t), \mathsf{s}_2(t), \ldots, \mathsf{s}_n(t)\rangle \mid (\lambda, t \models \mathsf{c})$. One timestamp and all the corresponding signal values where condition c is satisfied.
`d_not_becomes`$=\langle t, \mathsf{s}(t)\rangle \mid t \in (t_l, t_u], (\mathsf{s}(t) \sim \mathsf{v} \wedge \forall t_1 \in [t_l, t), (\mathsf{s}(t_1) \not\sim \mathsf{v}))$. The first record that satisfies $\mathsf{s}(t) \sim \mathsf{v}$, such that $\mathsf{s}(t_1) \not\sim \mathsf{v}$ for any time t_1 before t.
`d_not_spike` $= \langle(t_1, \mathsf{s}(t_1)), (t_5, \mathsf{s}(t_5))\rangle \mid \exists t_2, t_3, t_4 \in [t_l, t_u], (t_l < t_1 < t_2 < t_3 < t_4 < t_5 \wedge \mathit{uni\_m\_max}(\mathsf{s}, t_2, [t_1, t_3]) \wedge \mathit{uni\_sm\_min}(\mathsf{s}, t_3, [t_2, t_4]) \wedge \mathit{uni\_m\_max}(\mathsf{s}, t_4, [t_3, t_5])[[\wedge(t_3 - t_1) \sim_1 \mathsf{v}_1]_\beta[\wedge \max((\mathsf{s}(t_2) - \mathsf{s}(t_3)), (\mathsf{s}(t_4) - \mathsf{s}(t_3))) \sim_2 \mathsf{v}_2]_\gamma]_\alpha)$. The first $\langle t_1, \mathsf{s}(t_1)\rangle$ and the last $\langle t_5, \mathsf{s}(t_5)\rangle$ records that show an occurrence of a spike*.
`d_not_oscillation` $= \langle(t_1, \mathsf{s}(t_1)), (t_5, \mathsf{s}(t_5))\rangle \mid \exists t_2, t_3, t_4 \in [t_l, t_u], (t_1 < t_2 < t_2 < t_3 < t_4 < t_5 \wedge \mathit{uni\_sm\_max}(\mathsf{s}, t_2, [t_1, t_3]) \wedge \mathit{uni\_sm\_min}(\mathsf{s}, t_3, [t_2, t_4]) \wedge \mathit{uni\_sm\_max}(\mathsf{s}, t_4, [t_3, t_5])[[\wedge(t_4 - t_2) \sim_1 \mathsf{v}_1]_\zeta[\wedge(\mathsf{s}(t_2) - \mathsf{s}(t_3)) \sim_2 \mathsf{v}_2 \wedge (\mathsf{s}(t_4) - \mathsf{s}(t_3)) \sim_2 \mathsf{v}_2]_\epsilon]_\delta)$. The first $\langle t_1, \mathsf{s}(t_1)\rangle$ and the last $\langle t_5, \mathsf{s}(t_5)\rangle$ records that show an occurrence of oscillations*.
`d_not_rises` $= \langle t, \mathsf{s}(t)\rangle \mid t \in (t_l, t_u], (\mathsf{s}(t) \geq \mathsf{v} \wedge \forall t_1 \in [t_l, t), (\mathsf{s}(t_1) < \mathsf{v})[\wedge \mathit{mon}(\mathsf{s}, t_l, t)]_\alpha)$. The first record $\langle t, \mathsf{s}(t)\rangle$ at which the signal becomes greater than or equal to v, where the optional monotonicity constraint is satisfied, if defined in the property.
`d_not_overshoots` $= \langle t, \mathsf{s}(t)\rangle \mid t \in (t_l, t_u], (\mathsf{s}(t) \geq \mathsf{v}_1 \wedge \forall t_1 \in [t, t_u], (\mathsf{s}(t_1) \leq \mathsf{v}_1 + \mathsf{v}_2) \wedge \forall t_2 \in [t_l, t), (\mathsf{s}(t_2) < \mathsf{v}_1)[\wedge \mathit{mon}(\mathsf{s}, t_l, t)]_\alpha)$. The first record at which signal s reaches value v_1. The signal never goes above the maximum allowed amplitude of $\mathsf{v}_1 + \mathsf{v}_2$ and satisfies the monotonicity constraint, if defined in the property.
`d_not_if-then` $= \langle [t_1, t_2], [t_3, t_4]\rangle \mid (t_l < t_1 < t_2 < t_3 < t_4 < t_u \wedge \lambda, [t_1, t_2] \models \mathsf{p}_1 \wedge \lambda, [t_3, t_4] \models \mathsf{p}_2[\wedge(t_3 - t_2)[\![\bowtie]\!]\mathsf{d}]_\alpha)$. An interval $[t_1, t_2]$ where pattern p_1 holds and a subsequent interval $[t_3, t_4]$ where pattern p_2 holds.

`d_a_at`$_1$/**`d_a_bef`**$_1$/**`d_a_aft`**$_1$ $= \langle [t_i, t_e], t\rangle$. The time interval $[t_i, t_e]$ and the absolute boundary t, that is not within that interval.
`d_a_bet`$_1$ $= \langle [t_i, t_e], n, m\rangle$. Values n and m and the interval $[t_i, t_e]$.
`d_e_bef`$_1$ $= \langle [t_1, t_2]\rangle \mid (t_l < t_1 < t_2 \leq t_u \wedge \lambda, [t_1, t_2] \models p_1 \wedge \forall t_3, t_4, (t_l \leq t_3 < t_4 < t_1 \Rightarrow \lambda, [t_3, t_4] \not\models p))$. The interval $[t_1, t_2]$ where p_1 holds, and before which the property pattern p failed to hold.
`d_e_aft`$_1$ $= \langle [t_1, t_2]\rangle \mid (t_l \leq t_1 < t_2 < t_u \wedge \lambda, [t_1, t_2] \models p_1 \wedge \forall t_3, t_4, (t_2 < t_3 < t_4 \leq t_u \Rightarrow \lambda, [t_3, t_4] \not\models p))$. The interval $[t_1, t_2]$, where p_1 holds and after which the property pattern p failed to hold.
`d_e_bet`$_1$ $= \langle [t_2, t_3]\rangle \mid (\exists t_1 \in [t_l, t_u), t_l \leq t_1 < t_2 < t_3 < t_u \wedge \lambda, [t_1, t_2] \models p_1 \wedge \exists t_4 \in (t_3, t_u], t_3 < t_4 \leq t_u \wedge \lambda, [t_3, t_4] \models p_2 \wedge \lambda, [t_2, t_3] \not\models p)$. The time interval $[t_2, t_3]$, where t_2 is the last timestamp in which pattern p_1 held and t_3 is the first timestamp in which pattern p_2 held.

Figure 8. Diagnoses associated with the violation causes of atoms and scopes in Figure 7.

to represent a timestamp and a signal value observed in that timestamp. Therefore, the corresponding violation cause **`c_assert`**$_1$ checks for the presence of a timestamp in which the assertion condition c is violated.

For example, the trace shown in Figure 10 violates the expression "**`assert`** $\beta_1 < 4$" because signal β_1 shows a value equal to 5 at timestamp 4, satisfying the violation cause **`c_assert`**$_1$ on the interval $[0, 7]$.

Diagnoses: The diagnosis **`d_assert`**$_1$ associated with violation cause **`c_assert`**$_1$ includes the first timestamp t at which one or more signals $(\mathsf{s}_1, \mathsf{s}_2, \ldots, \mathsf{s}_n)$ violate the assertion condition c, as well as the values taken by these signals at t. This diagnosis allows engineers to identify the root cause of the violation of the assertion condition by looking at the first timestamp in which this violation was observed.

For instance, in the case of the trace shown in Figure 10, the diagnosis is the tuple containing timestamp 4 and the value of $\beta_1(4) = 5$ taken by signal β_1.

6.1.2 *`becomes`*: *State-based Data Assertion*

Violation causes: This pattern can be violated in at least three ways, as illustrated with different signal behaviors in Figure 11 using the expression "β **`becomes`** > 3":

- **`c_becomes`**$_1$: The signal value violates the pattern constraint $\sim \mathsf{v}$ throughout the time interval over which the pattern is evaluated. For instance, signal β_1 in the figure is never greater than 3.
- **`c_becomes`**$_2$: The signal value satisfies the pattern constraint $\sim \mathsf{v}$ throughout the time interval over which the pattern is evaluated. This violation cause is the dual of the previous case. For instance, signal β_2 in the figure is always greater than value 3.
- **`c_becomes`**$_3$: The signal violates the semantics of the pattern by satisfying the negation of the pattern constraint (i.e., $\lambda, [t_l, t_u] \models \mathsf{s}$ **`becomes`** $\not\sim \mathsf{v}$ holds).
  For instance, signal β_3 in the figure becomes less than or equal to 3 (instead of becoming greater than 3). More precisely, it goes below value 3 at timestamps 4, and remains below that value until the end of the time interval, delimited by timestamp 7.

Diagnoses: The diagnoses associated with the three violation causes above are the following:

- **`d_becomes`**$_1$ and **`d_becomes`**$_2$ include two records from the signal showing a minimum and a maximum value. In this way, we show the range of values over which the signal changes. In the example shown in Figure 11, we report records $\langle(7, 2.8), (4, 0.5)\rangle$ for signal β_1 and records $\langle(0.5, 5), (5, 3.3)\rangle$ for signal β_2.
- **`d_becomes`**$_3$ includes the last-seen record at which the signal value satisfies the constraint $\sim \mathsf{v}$, followed by the next-seen record at which the signal value satisfies $\not\sim \mathsf{v}$. Through this diagnosis, we want to capture the exact time interval, delimited by two consecutive timestamps, within $[t_l, t_u]$, in which the signal exhibits a behavior compatible with the negation of the constraint specified in the **`becomes`** expression. For instance, for signal β_3 in Figure 11, the diagnosis is $\langle(3, 4.3), (4, 0.8)\rangle$.

6.1.3 *Spike*

Violation causes: This pattern can be violated in at least five ways, as illustrated with different signal behaviors in Figure 5 using the expression "**`exists spike in`** β **`with amplitude`** < 90 **`width`** < 0.5". These alternatives are the following:

- **`c_spike`**$_1$: All spike instances in the signal violate the amplitude constraint. For instance, signal β_1 in the figure shows two spike amplitude values greater than 90 (150 and 200, respectively).
- **`c_spike`**$_2$: All spike instances in the signal violate the width constraint. For example, signal β_1 shows two spike width values greater than 0.5 (1.8 and 4.2, respectively).
- **`c_spike`**$_3$: The signal is constant throughout the time interval over which the pattern is evaluated. For exam-

d_assert$_1$ $= \langle t, \mathsf{s}_1(t), \mathsf{s}_2(t), \dots, \mathsf{s}_n(t)\rangle \mid (\lambda, t \not\models \mathsf{c}) \wedge \forall t_1 \in [t_l, t), (\lambda, t_1 \models \mathsf{c})$. The first timestamp t and the values, taken in correspondence of t, of the signals that lead to the violation of condition c.

d_becomes$_1$/**d_becomes**$_2$ $= \langle (t_1, \mathsf{s}(t_1)), (t_2, \mathsf{s}(t_2))\rangle \mid \forall t \in [t_l, t_u], (\mathsf{s}(t) \leq \mathsf{s}(t_1) \wedge \mathsf{s}(t) \geq \mathsf{s}(t_2))$. The maximum and the minimum values (and the corresponding timestamps) of signal s.

d_becomes$_3$ $= \langle (t_1, \mathsf{s}(t_1)), (t_2, \mathsf{s}(t_2))\rangle \mid t_l \leq t_1 < t_2 \leq t_u \wedge \mathsf{s}(t_1) \sim \mathsf{v} \wedge \mathsf{s}(t_2) \not\sim \mathsf{v} \wedge \neg\exists t_3 \in [t_l, t_u], (t_1 < t_3 < t_2)$. The last time instant t_1 (and the corresponding value) at which the signal s satisfies the predicate $\mathsf{s}(t_1) \sim \mathsf{v}$, exactly followed by the next time instant t_2 (and the corresponding value) at which the signal value satisfies the predicate $\mathsf{s}(t_2) \not\sim \mathsf{v}$.

d_spike$_1$ $= \langle [t_1, t_2], a\rangle \mid (\exists t_3, t_4, t_5 \in [t_l, t_u], (spk(\mathsf{s}, t_3, t_1, t_4, t_2, t_5) \wedge \neg(amp(\mathsf{s}, t_1, t_4, t_2) \sim_2 \mathsf{v}_2) \wedge a = amp(\mathsf{s}, t_1, t_4, t_2) \wedge \forall t_6, t_7, t_8, t_9, t_{10} \in [t_l, t_u], ((t_7 \neq t_1 \wedge t_8 \neq t_4 \wedge t_9 \neq t_2 \wedge spk(\mathsf{s}, t_6, t_7, t_8, t_9, t_{10})) \Rightarrow |a - \mathsf{v}_2| < ampv(\mathsf{s}, t_7, t_8, t_9, \mathsf{v}_2))))$. The amplitude a and the interval $[t_1, t_2]$ of the spike that is the closest to satisfy the amplitude constraint.

d_spike$_2$ $= \langle [t_1, t_2], w\rangle \mid (\exists t_3, t_4, t_5 \in [t_l, t_u], (spk(\mathsf{s}, t_3, t_1, t_4, t_2, t_5) \wedge \neg(width(t_1, t_2) \sim_1 \mathsf{v}_1) \wedge w = width(t_1, t_2) \wedge \forall t_6, t_7, t_8, t_9, t_{10} \in [t_l, t_u], ((t_7 \neq t_1 \wedge t_8 \neq t_4 \wedge t_9 \neq t_2 \wedge spk(\mathsf{s}, t_6, t_7, t_8, t_9, t_{10})) \Rightarrow |w - \mathsf{v}_1| < widthv(t_7, t_9, \mathsf{v}_1))))$. The width w and the time interval $[t_1, t_2]$ of the spike that is the closest to satisfy the width constraint.

d_spike$_3$ $= \langle [t_l, t_u], s(t_l)\rangle$. The first and the last timestamps (t_l and t_u) delimiting the interval throughout which signal s is constant, and the signal value.

d_spike$_4$/**d_spike**$_5$ $= \langle (t_1, \mathsf{s}(t_1)), (t_2, \mathsf{s}(t_2))\rangle \mid \forall t \in [t_l, t_u], (\mathsf{s}(t) \leq \mathsf{s}(t_1) \wedge \mathsf{s}(t) \geq \mathsf{s}(t_2))$. The maximum and the minimum values (and their timestamps) taken by signal s.

d_oscillation$_1$ $= \langle [t_1, t_5], a\rangle \mid (\exists t_2, t_3, t_4 \in [t_l, t_u], (osc(\mathsf{s}, t_1, t_2, t_3, t_4, t_5) \wedge \neg(p2p(\mathsf{s}, t_2, t_3) \sim_2 \mathsf{v}_2 \vee p2p(\mathsf{s}, t_3, t_4) \sim_2 \mathsf{v}_2) \wedge a = \max(p2p(\mathsf{s}, t_2, t_3), p2p(\mathsf{s}, t_3, t_4)) \wedge \forall t_6, t_7, t_8, t_9, t_{10} \in [t_l, t_u], ((t_8 \neq t_2 \wedge t_9 \neq t_3 \wedge t_{10} \neq t_4 \wedge osc(\mathsf{s}, t_6, t_8, t_9, t_{10}, t_7)) \Rightarrow (|a - \mathsf{v}_2| \leq p2pv(\mathsf{s}, t_8, t_9, \mathsf{v}_2) \wedge |a - \mathsf{v}_2| \leq p2pv(\mathsf{s}, t_9, t_{10}, \mathsf{v}_2)))))$. The amplitude a and the time interval $[t_1, t_5]$ of the closest oscillation instance to satisfy the amplitude constraint.

d_oscillation$_2$ $= \langle [t_1, t_5], w\rangle \mid (\exists t_2, t_3, t_4 \in [t_l, t_u], (osc(\mathsf{s}, t_1, t_2, t_3, t_4, t_5) \wedge \neg(width(t_2, t_4) \sim_1 \mathsf{v}_1) \wedge w = width(t_2, t_4) \wedge \forall t_6, t_7, t_8, t_9, t_{10} \in [t_l, t_u], ((t_8 \neq t_2 \wedge t_9 \neq t_3 \wedge t_{10} \neq t_4 \wedge osc(\mathsf{s}, t_6, t_8, t_9, t_{10}, t_7)) \Rightarrow (|w - \mathsf{v}_1| < widthv(t_8, t_{10}, \mathsf{v}_1)))))$. The period w and the interval $[t_1, t_2]$ of the closest oscillations instance to satisfy the period constraint.

d_oscillation$_3$ $= \langle t_1, \mathsf{s}(t_1)\rangle \mid \exists t_2, t_3 \in [t_l, t_u], t_l \leq t_2 < t_1 \wedge t_1 < t_3 \leq t_u \wedge (uni\_sm\_min(\mathsf{s}, t_1, [t_2, t_3]) \vee uni\_sm\_max(\mathsf{s}, t_1, [t_2, t_3]))$. The record at which the only seen strict extremum occurs in the signal, within the time interval $[t_l, t_u]$.

d_oscillation$_4$ $= \langle (t_1, \mathsf{s}(t_1)), (t_4, \mathsf{s}(t_4))\rangle \mid \exists t_2, t_3 \in [t_l, t_u], t_l \leq t_2 < t_1 \wedge t_1 < t_4 \wedge t_4 < t_3 \leq t_u \wedge (uni\_sm\_min(\mathsf{s}, t_1, [t_2, t_4]) \vee uni\_sm\_max(\mathsf{s}, t_1, [t_2, t_4])) \wedge (uni\_sm\_min(\mathsf{s}, t_4, [t_1, t_3]) \vee uni\_sm\_max(\mathsf{s}, t_4, [t_1, t_3])) \wedge t_4 \neq t_1$. The two records at which the strict maximum and the strict minimum occur in the signal, within the time interval $[t_l, t_u]$.

d_oscillation$_5$ $= \langle [t_l, t_u], \mathsf{s}(t_l)\rangle$. The first and the last timestamps (t_l and t_u) delimiting the interval throughout which signal s is constant, and the signal value.

d_oscillation$_6$/**d_oscillation**$_7$ $= \langle (t_1, \mathsf{s}(t_1)), (t_2, \mathsf{s}(t_2))\rangle \mid \forall t \in [t_l, t_u], (\mathsf{s}(t) \leq \mathsf{s}(t_1) \wedge \mathsf{s}(t) \geq \mathsf{s}(t_2))$. The maximum and the minimum values (and their timestamps) taken by the signal s.

d_rises$_1$/**d_rises**$_2$ $= \langle (t_1, \mathsf{s}(t_1)), (t_2, \mathsf{s}(t_2))\rangle \mid \forall t \in [t_l, t_u], (\mathsf{s}(t) \leq \mathsf{s}(t_1) \wedge \mathsf{s}(t) \geq \mathsf{s}(t_2))$. The maximum and the minimum values (and their timestamps) of signal s.

d_rises$_3$ $= \langle (t_1, \mathsf{s}(t_1)), (t_2, \mathsf{s}(t_2))\rangle \mid \neg(\exists t \in [t_l, t_u], (t_1 < t < t_2)) \wedge \exists t \in (t_l, t_u], (\mathsf{s}(t) \geq \mathsf{v} \wedge \forall t_3 \in [t_l, t), (\mathsf{s}(t3) < v) \wedge t_1 < t_2 < t \wedge \mathsf{s}(t_1) > \mathsf{s}(t_2))$. Two signal values that violate the monotonicity constraint and the corresponding consecutive timestamps t_1 and t_2.

d_rises$_4$ $= \langle (t_1, \mathsf{s}(t_1)), (t_2, \mathsf{s}(t_2))\rangle \mid t_l \leq t_1 < t_2 \leq t_u \wedge \mathsf{s}(t_1) \geq \mathsf{v} \wedge \mathsf{s}(t_2) < \mathsf{v} \wedge \neg\exists t \in [t_l, t_u], (t_1 < t < t_2)$. The record at which the signal s is greater than or equal to value v, followed by the record at which the signal falls, going below v.

d_overshoots$_1$/**d_overshoots**$_2$ $= \langle (t_1, \mathsf{s}(t_1)), (t_2, \mathsf{s}(t_2\rangle)) \mid \forall t \in [t_l, t_u], (\mathsf{s}(t) \leq \mathsf{s}(t_1) \wedge \mathsf{s}(t) \geq \mathsf{s}(t_2))$. The maximum and the minimum values (and timestamps) of signal s.

d_overshoots$_3$ $= \langle t_1, \mathsf{s}(t_1), t_2, \mathsf{s}(t_2)\rangle \mid \neg(\exists t \in [t_l, t_u], (t_1 < t < t_2)) \wedge \exists t \in (t_l, t_u], (\mathsf{s}(t) \geq \mathsf{v}_1 \wedge \forall t_4 \in [t_l, t), (\mathsf{s}(t_4) \leq \mathsf{v}_1) \wedge \forall t_5 \in [t, t_u], (\mathsf{s}(t_5) \leq \mathsf{v}_1 + \mathsf{v}_2) \wedge (t_l < t_1 < t_2 < t) \wedge \mathsf{s}(t_1) \geq \mathsf{s}(t_2))$. Two consecutive records of a signal that overshoots, but does not satisfy the monotonicity constraint.

d_overshoots$_4$ $= \langle (t_1, \mathsf{s}(t_1)), (t_2, \mathsf{s}(t_2)\rangle \mid t_l \leq t_1 < t_2 \leq t_u \wedge \mathsf{s}(t_1) \geq \mathsf{v}_1 \wedge \mathsf{s}(t_1) \leq \mathsf{v}_1 + \mathsf{v}_2 \wedge \mathsf{s}(t_2) < \mathsf{v}_1 \wedge \neg\exists t \in [t_l, t_u], (t_1 < t < t_2)$. The record at which the signal s is greater than or equal to value v_1 and less than or equal to v_1+v_2, followed by the record at which the signal undershoots, going below v_1.

d_if-then$_1$ $= \langle [t_2, t_u]\rangle \mid (\exists t_1 \in [t_l, t_2), \lambda, [t_1, t_2] \models \mathsf{p}_1 \wedge \forall t_3, t_4 \in (t_2, t_u], (\lambda, [t_3, t_4] \not\models \mathsf{p}_2))$. The time interval delimited by t_2 (the last time instant of the last occurrence of pattern p_1) up to the last time instant (t_u) of the trace.

d_if-then$_2$ $= \langle [t_2, t_3], t_3 - t_2\rangle \mid (\exists t_1 \in [t_l, t_2), \lambda, [t_1, t_2] \models \mathsf{p}_1 \wedge \exists t_4 \in (t_3, t_u], (\lambda, [t_3, t_4] \models \mathsf{p}_2 \Rightarrow \neg((t_3 - t_2) [\![\bowtie]\!] \mathsf{d}) \wedge \forall t_5, t_6 \in (t_2, t_3), (\lambda, [t_5, t_6] \not\models \mathsf{p}_2)))$. The time interval $[t_2, t_3]$ representing the time distance between patterns p_1 and p_2 hold, and the exact value of that violated time distance $(t_3 - t_2)$.

* $spk(\mathsf{s}, t_1, t_2, t_3, t_4, t_5) = uni\_m\_min(\mathsf{s}, t_2, [t_1, t_3]) \wedge uni\_sm\_max(\mathsf{s}, t_3, [t_2, t_4]) \wedge uni\_m\_min(\mathsf{s}, t_4, [t_3, t_5])$
* $osc(\mathsf{s}, t_1, t_2, t_3, t_4, t_5) = uni\_sm\_min(\mathsf{s}, t_2, [t_1, t_3]) \wedge uni\_sm\_max(\mathsf{s}, t_3, [t_2, t_4]) \wedge uni\_sm\_min(\mathsf{s}, t_4, [t_3, t_5])$
$ampv(\mathsf{s}, t_1, t_2, t_3, \mathsf{v}) = |amp(\mathsf{s}, t_1, t_2, t_3) - \mathsf{v}|$; $p2pv(\mathsf{s}, t_1, t_2, \mathsf{v}) = |p2p(\mathsf{s}, t_1, t_2) - \mathsf{v}|$; $widthv(t_1, t_2, \mathsf{v}) = |width(t_1, t_2) - \mathsf{v}|$

Figure 9. Diagnoses associated with the violation causes of patterns in Figure 7.

ple, the constant signal β_2 in the figure always takes the value 100 within the time interval $[0, 6]$.

- **c_spike**$_4$: The signal decreases,within the time interval over which the pattern is evaluated, without showing any spike behavior. For example, signal β_3 in the figure decreases within the time interval $[0, 6]$, going from value 190 to 30.
- **c_spike**$_5$: The signal increases within the time interval over which the pattern is evaluated, without showing any spike behavior. For instance, signal β_4 increases within the time interval $[0, 6]$, going from value 30 to 190.

Diagnoses: The diagnoses associated with the five violation causes above are the following:

- **d_spike**$_1$ includes the time interval in which the spike with the closest amplitude to satisfy the amplitude constraint occurs, as well as the amplitude value of that spike instance (see page 6 for a detailed explanation). The intuition behind this diagnosis is that when a spike property with an amplitude constraint is violated, the engineers are interested in knowing the amplitude value of the spike that is the closest to satisfy the amplitude constraint, to assess how close the signal behavior was to satisfy the property . For instance, for signal β_1 in Figure 5, the diagnosis is $\langle [0, 1.8], 150\rangle$.
- **d_spike**$_2$ is defined in a similar way, but with respect to the width constraint. It includes the time interval in which the spike with the closest width to satisfy the

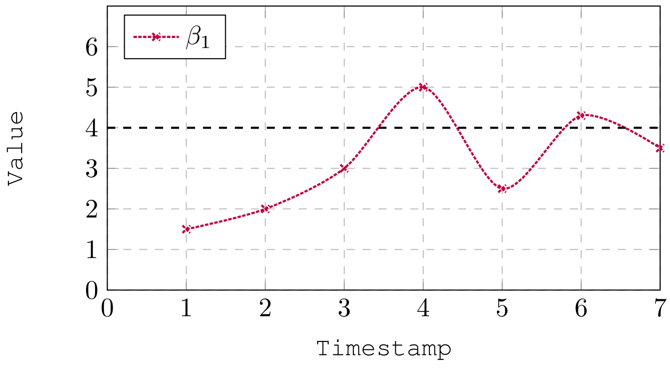


Figure 10. A trace violating the expression "**`assert`** $\beta < 4$".

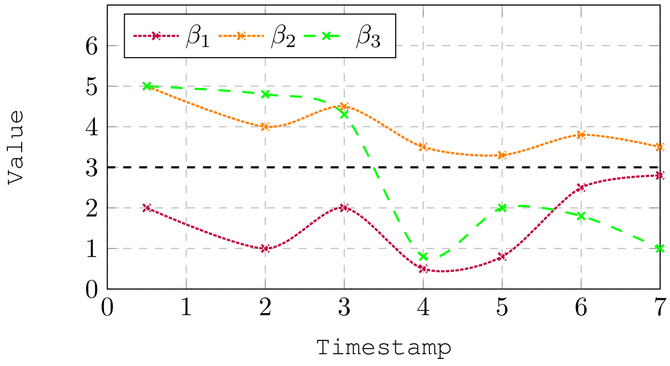


Figure 11. A trace with signals violating the expression "β **`becomes`** $>$ 3".

width constraint occurs, as well as the width value of that spike instance. Similar to **`d_spike`**$_1$, the choice of this specific width value enables engineers to determine the closest value of a spike width to the satisfaction of the width constraint defined in the pattern. For instance, for signal β_1 in Figure 5, the diagnosis is $\langle[0, 1.8], 1.8\rangle$.

- **`d_spike`**$_3$ includes the time interval $[t_l, t_u]$ over which the property pattern is evaluated, as well as the value taken by the constant signal throughout that interval. This diagnosis shows that the signal is constant (i.e., it shows a single value) throughout the full time interval $[t_l, t_u]$, over which the pattern is evaluated. For instance, for signal β_2 in Figure 5, the diagnosis is $\langle[0, 6], 100\rangle$.
- **`d_spike`**$_4$ and **`d_spike`**$_5$ include two records from the signal corresponding to its minimum and maximum values (and the timestamps at which these values occur). In this way, we show the range of values over which the signal changes (decreasing or increasing). In the example shown in Figure 5, we report records $\langle(0, 200), (6, 55)\rangle$ for the decreasing signal β_3 and records $\langle(0, 30), (6, 190)\rangle$ for the increasing signal β_4.

6.1.4 Oscillation

Violation causes: This pattern can be violated in at least seven ways, as illustrated with different signal behaviors in Figure 12 using the expression "**`exist oscillation in`** β **`with p2pAmp`** < 90 **`period`** < 0.5". These alternatives are the following:

- **`c_oscillation`**$_1$: All oscillation instances in the signal violate the amplitude constraint. For instance, signal β_1 in the figure shows two oscillation instances, both having an amplitude value greater than 90 (125 and 200, respectively).
- **`c_oscillation`**$_2$: All oscillation instances in the signal violate the period constraint. For instance, signal β_1 shows two oscillation instances whose period value is greater than 0.5: the first oscillation has a period of 0.8 (i.e., the time difference between timestamps 0.2 and 1), while the second oscillation has a period of 1 (i.e., the time difference between timestamps 3.5 and 4.5).
- **`c_oscillation`**$_3$: The signal does not show any oscillation; instead, it shows only one strict local extremum (a maximum or a minimum). This is the case, for instance, of signal β_2 in the figure, that exhibits a strict local maximum (reaching the value of 150 at timestamp 1.5).
- **`c_oscillation`**$_4$: The signal does not show any oscillation; instead, it shows only two strict local extrema. For instance, signal β_3 in the figure exhibits a strict local minimum (taking value 80 at timestamp 1.5), followed by a strict local maximum (taking value 150 at timestamp 2).
- **`c_oscillation`**$_5$: The signal is constant throughout the time interval $[t_l, t_u]$ (see, for example, signal β_4 in the figure).
- **`c_oscillation`**$_6$: The signal decreases without showing any oscillatory behavior. For instance, signal β_5 in the figure decreases, going from value 180 at timestamp 0.1 to value 20 at timestamp 5.8.
- **`c_oscillation`**$_7$: The signal increases without showing any oscillatory behavior. For example, signal β_6 in the figure increases, going from value 40 at timestamp 0.2 to value 150 at timestamp 5.8.

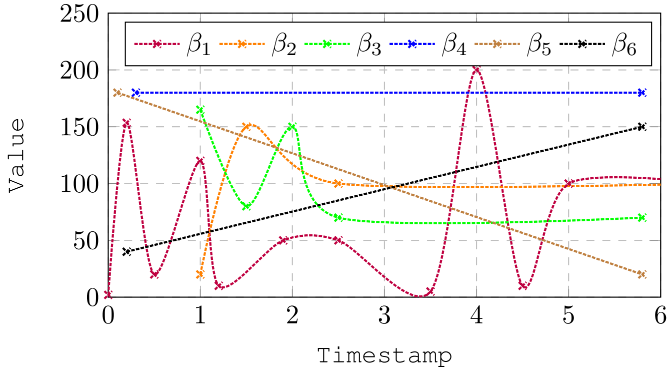


Figure 12. A trace with signals violating the expression "**`exist oscillation in`** β **`with p2pAmp`** < 90 **`period`** < 0.5 ".

Diagnoses: The diagnoses associated with the seven violation causes above are the following:

- **`d_oscillation`**$_1$ includes the time interval in which the oscillation with the closest amplitude to satisfy the amplitude constraint occurs, as well as the amplitude value of that oscillation instance. The choice of this diagnosis enables engineers to determine the oscillation instance with the closest amplitude to the satisfaction of the amplitude constraint defined in the pattern. For instance, for signal β_1 in Figure 12, the diagnosis is $\langle [0, 1.9], 125\rangle$.

- **d_oscillation**$_2$ includes the time interval in which the oscillation with the closest period to satisfy the period constraint occurs, as well as the period of that oscillation instance. For instance, for signal β_1 in Figure 12, the diagnosis is $\langle [0, 1.9], 0.8\rangle$. Similar to **d_oscillation**$_1$, we allow engineers to identify the oscillation instance that shows the closest period value to the satisfaction of the period constraint defined in the pattern.
- **d_oscillation**$_3$ includes the timestamp (and the corresponding signal value) in which the signal exhibits a strict extremum. The reported diagnosis allows engineers to identify the first time in which the signal exhibited a considerable deviation, leading to a change of the sign of its derivative. For instance, for signal β_2 in Figure 12, the diagnosis is $\langle 1.5, 150\rangle$.
- **d_oscillation**$_4$ includes the two records from the signal in which the strict maximum and the strict minimum occur. By considering this diagnosis, engineers are able to see a considerable change of the signal shape, showing two different consecutive strict extrema. For instance, for signal β_3 in Figure 12, the diagnosis is $\langle (1.5, 80), (2, 150)\rangle$.
- **d_oscillation**$_5$ includes the time interval $[t_l, t_u]$ throughout which the signal s is constant, as well as the value taken by that signal. Similar to **d_spike**$_3$, this diagnosis shows that the signal is constant throughout the full time interval $[t_l, t_u]$ over which the pattern is evaluated. For instance, for signal β_4 in Figure 12, the diagnosis is $\langle [0, 6], 180\rangle$.
- **d_oscillation**$_6$ and **d_oscillation**$_7$ include the records in which the maximum and the minimum values of the signal were observed. In this way, we show the range of values over which the signal changes (i.e., decreases or increases). In the example shown in Figure 12, we report records $\langle (0.1, 180), (5.8, 20)\rangle$ for signal β_5 and records $\langle (0.2, 40), (5.8, 150)\rangle$ for signal β_6.

6.1.5 Rise time

Violation causes: This pattern can be violated in at least four ways, as illustrated with different signal behaviors in Figure 13 using the expression "β **rises monotonically reaching 3**".

- **c_rises**$_1$: The signal is always below the threshold value v defined in the pattern constraint. For instance, signal β_1 is always below the value of 3, showing values ranging between 0.8 and 2.5.
- **c_rises**$_2$: The signal is always greater than or equal to the threshold value v. For instance, signal β_2 is always above the value 3, showing values ranging between 4 and 6.
- **c_rises**$_3$: The signal shows a rising behavior, but violates the monotonicity constraint defined in the pattern. For instance, signal β_3 rises reaching the target value (showing a value of 4 at timestamp 4), but it violates the monotonicity constraint since its value decreases from 2 (at timestamp 2) to 0.5 (at timestamp 3).
- **c_rises**$_4$: The signal is initially above the threshold value v. It then falls (and remains) below that value, instead of rising. For instance, signal β_4 falls (and remains) below the target value of 3 (starting from timestamp 4, up to timestamp 7, showing values ranging within the interval $[0.5, 2]$) instead of rising.

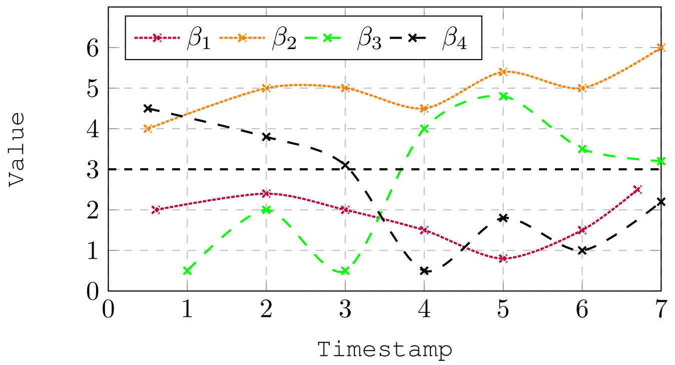


Figure 13. A trace with signals violating the expression "β **rises monotonically reaching** 3".

Diagnoses: The diagnoses associated with the four violation causes above are the following:

- **d_rises**$_1$ and **d_rises**$_2$ include the records in which the signal shows a maximum and a minimum value. In this way, we show the range of values the signal takes. For instance, for signal β_1 in Figure 13, the diagnosis is $\langle (5, 0.8), (6.7, 2.5)\rangle$. Similarly, diagnosis for signal β_2 is $\langle (0.5, 4), (7, 6)\rangle$.
- **d_rises**$_3$ includes two consecutive records where the monotonicity constraint is violated. In this way, we show the exact interval over which the signal deviated from the last time it exhibited an increasing behavior, showing a negative derivative. For instance, for signal β_3 in Figure 13, the diagnosis is $\langle (2, 2), (3, 0.5)\rangle$.
- **d_rises**$_4$ includes two consecutive records in which the signal shows a dual behavior (i.e., it falls instead of rising). More precisely, the signal value in the first record is above the threshold value v. The signal value in the second reported record is, however, below that value. This diagnosis determines the interval over which the signal shows a dual behavior, within the time interval $[t_l, t_u]$ over which the pattern is evaluated. For instance, the diagnosis of signal β_4 in Figure 13 is $\langle (3, 3.1), (4, 0.5)\rangle$.

6.1.6 Overshoot

Violation causes: This pattern can be violated in at least four ways, as illustrated with different signal behaviors in Figure 14 using the expression "β **overshoots monotonically** 3 **by** 1":

- **c_overshoots**$_1$: The signal violates the pattern constraint, by always showing values below the threshold value v_1. For example, signal β_1 is always below the value of 3.
- **c_overshoots**$_2$: The signal goes beyond the maximum allowed value, which consists of the sum of the

target value v_1 and the maximum threshold value v_2 ($\mathsf{v}_1 + \mathsf{v}_2$), and remains above that value. For instance, signal β_2 exceeds 4 (showing a value of 4.5 at timestamp 2) and remains above the value of 4, ranging over $[4.1, 4.9]$.

- c_overshoots$_3$: The signal overshoots the threshold value v_1, without going beyond the maximum allowed value (delimited by $\mathsf{v}_1 + \mathsf{v}_2$ defined in the pattern). However, it violates the monotonicity constraint. For instance, signal β_3 overshoots, reaching the value of 3.8 at timestamp 4, without going beyond the value of 4 after then. It violates the monotonicity constraint within the time interval $[2, 3]$, since its value goes from 2 down to 0.5.
- c_overshoots$_4$: The signal shows a dual behavior: it undershoots, going below the value v_1, and remains below that value instead of overshooting. For instance, signal β_4 goes (and remains) below the value of 3. It reaches value 2 at timestamp 3 and takes, right after then, values ranging over $[0.5, 2]$.

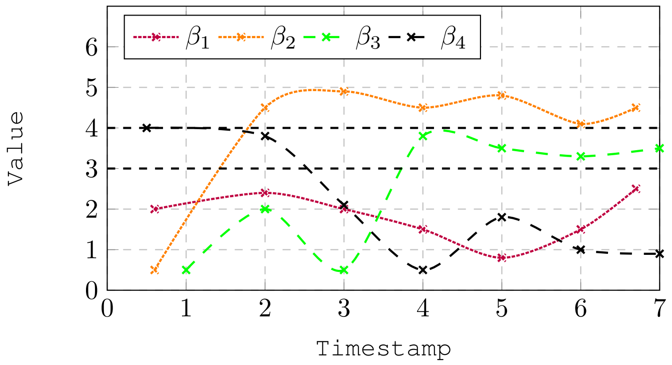


Figure 14. A trace with signals violating the expression "β overshoots monotonically 3 by 1".

Diagnoses: The diagnoses associated with the four violation causes above are the following:

- d_overshoots$_1$ and d_overshoots$_2$ include the records in which the signal shows a maximum and a minimum value. The reported diagnosis allows engineers to understand the range of values taken by the signal. For instance, for signal β_1 in Figure14, the diagnosis is $\langle(5, 0.8), (6.7, 2.5)\rangle$.
- d_overshoots$_3$ includes two consecutive records from a signal that overshoots, but violates the monotonicity constraint. This diagnosis shows the time interval over which the signal violated the monotonicity constraint within $[t_l, t_u]$. For instance, for signal β_3 in Figure14, the diagnosis is $\langle(2, 2), (3, 0.5)\rangle$.
- d_overshoots$_4$ includes two consecutive records in which the signal shows a dual behavior (i.e., it undershoots instead of overshooting). More specifically, we report the last record at which the signal value is delimited by $[\mathsf{v}_1, \mathsf{v}_1 + \mathsf{v}_2]$, followed by the next-seen record at which it undershoots, going below v_1. This diagnosis allows engineers to understand the interval over which the signal shows a dual behavior within the time interval $[t_l, t_u]$. For instance, for signal β_4 in Figure14, the diagnosis is $\langle(2, 3.8), (3, 2.1)\rangle$.

6.1.7 *Order relationship*

Violation causes: A property with an if-then construct is based on two patterns, each of which represents one of the pattern constructs we support in SB-TemPsy-DSL. According to the construct syntax if p_1 then p_2, p_1 is referred to as a cause pattern and p_2 as an effect pattern. We consider two possible violation causes of a property with the if-then construct.

- c_if-then$_1$: the cause pattern p_1 holds at some time interval $[t_1, t_2]$ within the time interval $[t_l, t_u]$, but then, the effect pattern p_2 fails to hold until the last timestamp (t_u) of that time interval.
- c_if-then$_2$: the cause pattern p_1 holds within a time interval $[t_1, t_2]$ but since then, whenever the effect pattern p_2 holds (after p_1) within a time interval $[t_3, t_4]$, the time distance ($t_3 - t_2$) between the occurrences of two patterns p_1 and p_2 is violated.

Diagnoses: The diagnoses associated with a violation of an expression with an if-then construct are the following:

- d_if-then$_1$ includes the time interval delimited by the last timestamp (t_2) of the last occurrence of pattern p_1 and the last timestamp t_u of the time interval $[t_l, t_u]$, showing the exact interval over which the effect pattern p_2 failed to hold. The corresponding diagnosis is then the following: $\langle[t_2, t_u]\rangle$.
- d_if-then$_2$ includes the time interval delimited by the last timestamp (t_2) in which the cause pattern p_1 holds, and the first timestamp (t_3) in which the effect pattern p_2 holds. The diagnosis also includes the violated time distance ($t_3 - t_2$) between the occurrence of patterns p_1 and p_2. The diagnosis is the following: $\langle[t_2, t_3], t_3 - t_2\rangle$.

6.2 *Scopes*

Since the same syntactic constructs of SB-TemPsy-DSL (e.g., the keyword before) can be used to define both absolute and event scopes (see Figure 3), we use the identifiers a_ and e_ before the violation cause name depending on whether it refers to an absolute or an event scope. Additionally, we use bef, aft, and bet as shortcuts for before, after, and between, respectively. For example, c_a_bef$_2$ denotes a violation cause for the before absolute scope.

Violation causes:

- c_a_at$_1$, c_a_bef$_1$ and c_a_aft$_1$ are violations related to absolute boundaries (i.e., timestamps) that are not within the time interval of the trace over which the property is evaluated.
- c_a_bet$_1$ indicates that either at least one of the scope boundaries is outside the time interval $[t_i, t_e]$ or the left boundary (which is supposed to be smaller than the right one) is greater than or equal to the right boundary.
- c_e_bef$_1$ states that scope pattern p_1 holds in the execution trace, whereas the property pattern p fails to hold sometime before p_1 held.
- c_e_aft$_1$ indicates that scope pattern p_1 holds in the execution trace, whereas the property pattern p fails to hold after that.
- c_e_bet$_1$ states that scope patterns p_1 and p_2 hold in the execution trace, whereas the property pattern p fails

to hold between the last timestamp where p_1 held and the first timestamp in which p_2 held.

Diagnoses: The diagnoses associated with scope-based violations are the following:

- **d_a_at**$_1$, **d_a_bef**$_1$ and **d_a_aft**$_1$ include the time interval $[t_i, t_e]$ that delimits the execution trace, as well as the absolute boundary t that is not within the range delimited by that time interval.
- **d_a_bet**$_1$ includes the time interval $[t_i, t_e]$ of the execution trace over which the property is evaluated, as well as the left and the right absolute boundaries of the scope (i.e., timestamps n and m, respectively; see Figure 3).
- **d_e_bef**$_1$ includes the time interval $[t_1, t_2]$ in which the scope pattern p_1 held and before which the property pattern p failed to hold.
- **d_e_aft**$_1$ includes the time interval $[t_1, t_2]$ in which the scope pattern p_1 held and after which the property pattern p failed to hold.
- **d_e_bet**$_1$ includes the time interval $[t_2, t_3]$ where t_2 represents the last timestamp in which pattern p_1 (the left event-boundary) held and t_3 (the right event-boundary) is the first timestamp in which pattern p_2 held throughout the execution trace.

6.3 *Atoms*

Violation causes: The violation cause **c_not**$_1$ for the construct "**not** sc" requires sc to be satisfied, since for "**not** sc" to be violated, sc must be satisfied (see the semantics in Figure 4).

Diagnosis: As depicted in Figure 8, many diagnosis are associated with the violation cause **c_not**$_1$. Indeed, for **c_not**$_1$ the diagnosis should explain why the violation cause **c_not**$_1$ (see Figure 7) holds, i.e., why $\mathtt{sc}$ is satisfied. The reasons that lead to the satisfaction of $\mathtt{sc}$ depend on the SB-TemPsy-DSL scope and the pattern used to define $\mathtt{sc}$. When $\mathtt{sc}$ is satisfied, both the scope and the pattern are satisfied. Our diagnosis explains *why the pattern used to define* sc *holds*. For this reason, the name of the diagnosis is obtained by adding the string "**d_not**" before the name of the pattern used to define $\mathtt{sc}$. In the following, we explain each of the diagnoses w.r.t the pattern defining $\mathtt{sc}$:

- diagnosis **d_not_assert** includes a timestamp t in which one or more signals $(s_1, s_2, \ldots, s_n)$ defined in the related property satisfy the corresponding condition c as well as the corresponding value(s) taken by each of these signals.
- diagnosis **d_not_becomes** includes the first timestamp t that satisfies the property condition, as well as the value of the signal recorded at t.
- diagnosis **d_not_spike** includes the first and the last records of a spike instance that occurred within the time interval $[t_l, t_u]$.
- diagnosis **d_not_oscillation** includes the first and the last records of an oscillation instance.
- diagnosis **d_not_rises** includes the first record at timestamp t in which (1) the signal defined in the property rises, reaching the property threshold v, and (2) the monotonicity constraint (if defined in the pattern) is satisfied within the time interval $[t_l, t]$.
- diagnosis **d_not_overshoots** includes the first record at which the signal defined in the property overshoots (i.e., reaching a value that ranges between values v and $\mathsf{v}_1 + \mathsf{v}_2$) and satisfies the monotonicity constraint, if defined in the pattern.
- diagnosis **d_not_if-then** includes two time intervals delimiting where the cause pattern and the effect one of the property hold throughout the execution trace.

7 *TD-SB-TemPsy* AT WORK

In this section, we illustrate how *TD-SB-TemPsy* works by applying algorithm 1 to three example properties, each of them with different constructs.

7.1 Property with a single atom

Let us consider property P1, checked on the trace shown in Figure 5:

$$\mathrm{P1} \equiv \quad \textbf{after}\ 7\ \textbf{exists spike in}\ \beta_1 \\ \textbf{with width} < 0.5\ \textbf{amplitude} < 90.$$

Based on the SB-TemPsy-DSL grammar in Figure 3, this property is made of a single atom of the form **after** t p, i.e., it consists of an **after** scope construct (delimited by an absolute time instant, parameter t = 7) constraining a pattern p of type **spike**.

Given the presence of only one atom in the property, algorithm 1 first determines whether the atom itself is violated by the trace (line 5). Since the trace violates the specification defined by the atom, algorithm 1 continues by computing the associated diagnosis using function TD-ATOM (algorithm 2).

Algorithm 2 relies on the auxiliary function GETVIOLATIONCAUSES, which analyzes the syntactic structure of the atom and determines the possible violation causes associated with it. In this case, the possible violation causes are the one associated with the **after** scope construct with an absolute boundary, i.e., **c_a_aft**$_1$, and the five ones associated with the **spike** construct, i.e., **c_spike**$_i$ with $1 \leq i \leq 5$. This means that function GETVIOLATIONCAUSES returns the list vcs = [**c_a_aft**$_1$, **c_spike**$_1$, **c_spike**$_2$, **c_spike**$_3$, **c_spike**$_4$, **c_spike**$_5$].

The algorithm continues by looping through the violation causes in vcs, to determine the first violation cause that holds on the trace; it will then return the corresponding diagnosis. In this example, the violation cause **c_a_aft**$_1$ holds on the trace since the value of parameter t (7) is outside the time interval $[0, 6]$. The corresponding diagnosis **d_a_aft**$_1$ = $\langle [0, 6], 7 \rangle$ shows the interval $[0, 6]$ and the absolute boundary 7.

7.2 Property with a single atom and negation

Let us consider property P2, checked on the trace shown in Figure 11:

$$\mathrm{P2} \equiv \quad \textbf{not globally}\ \beta_3\ \textbf{becomes} < 3$$

Based on the SB-TemPsy-DSL grammar in Figure 3, this property is made of a single atom of the form **not** sc, where sc ≡ **globally** p consists of a **globally** scope construct constraining a pattern p of type **becomes**.

As in the previous example, with only one atom in the property, algorithm 1 determines whether the atom itself is violated by the trace (line 5). Since the trace violates the specification defined by the atom, algorithm 1 continues by computing the associated diagnosis using function TD-ATOM (algorithm 2).

During the execution of algorithm 2, the auxiliary function GETVIOLATIONCAUSES returns the list $vcs = [$**c_not**$_1]$, since, in this example, the only possible violation cause is associated with the **not** construct.

Algorithm 2 will then compute the diagnosis corresponding to the only violation cause included in list vcs, using the auxiliary function GETDIAGNOSIS. In this example, the violation cause **c_not**$_1$ holds on the trace since there exists a time instant (timestamp 3) in which the value of signal β_3 decreases from value 4.5 to 0.9 (at timestamp 4). The corresponding diagnosis **d_not_becomes** $= \langle 4, 0.9\rangle$ shows the first record (at timestamp 4) in which the predicate associated with the **becomes** pattern holds, as well as the value of the signal.

7.3 Property with a conjunction of two atoms

Let us consider property P3, checked on the trace shown in Figure 13

P3 $\equiv$ **globally** β_3 **rises monotonically reaching** 3 **and between** 2 **and** 6 **assert** $\beta_3 <= 4$

Based on the SB-TemPsy-DSL grammar in Figure 3, this property is made of a single clause that consists of a conjunction of two atoms δ_1 and δ_2, where $\delta_1 \equiv$ **globally** β_3 **rises monotonically reaching** 3 and $\delta_2 \equiv$ **between** 2 **and** 6 **assert** $\beta_3 <= 4$. Atom δ_1 consists of a **globally** scope construct constraining a pattern p of type **rises**; atom δ_2 consists of a **between** scope construct (delimited by two absolute time instants, parameters $t_1 = 2$ and $t_2 = 6$) constraining a pattern p of type **assert**.

Given the presence of two atoms in the property, the loop at lines 4–6 of algorithm 1 is executed twice. More in details, algorithm 1 first determines whether atom δ_1 is violated by the trace. Since the trace violates the specification defined by the atom, algorithm 1 continues by computing the associated diagnosis using function TD-ATOM (algorithm 2). During the execution of the latter, the auxiliary function GETVIOLATIONCAUSES returns the list $vcs = [$**c_rises**$_1,$ **c_rises**$_2,$ **c_rises**$_3,$ **c_rises**$_4]$, since four possible violation causes are associated with the **rises** pattern construct. Function CHECKVIOLATIONCAUSE will then determine that the first violation cause (among those in vcs) that holds on the trace is **c_rises**$_3$, since signal β_3 violates the monotonicity constraint in two time instants (at timestamp 2 with value 2 and at timestamp 3 with value 0.5). The corresponding diagnosis, computed by function GETDIAGNOSIS, is **d_rises**$_3 = \langle\langle 2, 2\rangle, \langle 3, 0.5\rangle\rangle$, consisting of the tuples (each with a timestamp and the corresponding signal value) that violate the monotonicity constraint.

A similar process is followed for atom δ_2, which is also violated by the trace. In this case, function GETVIOLATIONCAUSES returns the list $vcs = [$**c_a_bet**$_1,$ **c_assert**$_1]$, since the possible violation causes are associated with the **between** scope construct and the **assert** pattern construct. The first violation cause that holds on the trace is **c_assert**$_1$, since signal β_3 violates the predicate associated with the assertion at timestamp 5, when its value reaches 4.9. The corresponding diagnosis **d_assert**$_1 = \langle 5, 4.9\rangle$ shows the timestamp and the signal value. Algorithm 1 then ends by returning the set of the diagnoses instances, containing **d_rises**$_3$ and **d_assert**$_1$.

8 EVALUATION

Recall that, in CPSs, temporal properties are often complex, since they are typically expressed as constraints on different signal behaviors. Although we support characterizations of individual signal behaviors, these can be and are often considered together, to report violations within a single property. As a result, we are interested in assessing the *applicability* of *TD-SB-TemPsy*, that is to which extent and how efficiently *TD-SB-TemPsy* is able to report diagnoses of industrial properties violated by industrial traces.

8.1 Datasets

To the best of our knowledge, there is no public dataset containing traces and properties suitable for investigating the diagnosis of signal-based temporal properties expressed in SB-TemPsy-DSL. For example, existing works on the topic of trace diagnostics, that use different specification languages (e.g., STL [13, 10]), have not released their traces and properties. Moreover, the lack of standardized benchmarks in the field of runtime verification is a well-known issue [26], hindered by the diversity of the tools' specification languages [27]. Existing specification-based generators for synthesized traces target a particular specification language, such as MFODL [28], MLTL [29], and MTL [30]. No trace generator exists for SB-TemPsy-DSL or for other languages for signal-based properties (like STL and HLS [31]).

In light of this, to investigate the applicability of *TD-SB-TemPsy*, we considered two different sources for obtaining traces and getting access to properties of interest:

- an industrial system from the satellite domain (hereafter referred to as PROP-SAT), provided by our industrial partner;
- the *Fuel Control of an Automotive Powertrain* (referred to as AFC) benchmark model [32] and its requirements used in the ARCH competition [19], a competition for the falsification of temporal logic specifications written in STL.

PROP-SAT dataset

This dataset was defined as follows. We considered 361 traces provided by our industrial partner; each of these traces logs the in-orbit operations of a satellite. The number of records in the traces ranges from $25\,358$ to $9\,328\,178$ ($avg = 438\,224$, $StdDev \approx 596\,505$), and the recording interval ranges from $25\,\mathrm{min}$ to $23\,\mathrm{h}.29\,\mathrm{min}$ ($avg = 6\,\mathrm{h}.38\,\mathrm{min}$, $StdDev = 7\,\mathrm{h}.5\,\mathrm{min}$).

We considered 98 properties defined with our industrial partner and expressed in SB-TemPsy-DSL. These properties were first elicited (and defined in English) through a series of meetings with a group of system and software engineers of our industrial partner. The corresponding SB-TemPsy-DSL properties were then written by the first author and validated by the engineers. This task cumulatively lasted about 80 hours.

The number of occurrences of each scope and pattern construct of SB-TemPsy-DSL in the properties is the following. For scopes: **`globally`** 73, **`before`** 1, **`after`** 8, **`at`** 3, **`between`** 15; for patterns: **`assert`** 111, **`becomes`** 13, **`spike`** 5, **`oscillation`** 23, **`rises`** 3, **`falls`** 7, **`overshoots`** 4, **`undershoots`** 4, **`if-then`** 23. All the listed scopes and pattern constructs are used in the definition of at least one property; we remark that none of the properties used the **`not`** `sc` construct for defining atoms.

We considered $361 \times 98 = 35\,378$ trace-property combinations, each obtained from one of the 361 traces and one of the 98 properties.

We removed $13\,426$ trace-property combinations for which the trace did not log (in any of its records) any variables used in the property and therefore did not enable the verification of its satisfaction. Such a situation occurred because some properties are supposed to be checked *only* at a certain operational stage (e.g., only during the launch phase and not during the operational phase). As a result, system engineers chose not to instrument the system, at certain stages of operation, when properties were not meant to be verified.

Then, we iteratively considered each of the remaining $21\,952$ trace-property combinations. Due to the sampling strategy used by our industrial partner, two records of the same trace may log different variables, i.e., a value may not be present in every record for some of the variables. Therefore, to ensure that the traces have the format described in section 2, we proceeded as follows. For each trace-property combination, we (a) removed entirely from the trace all the records that only contain variable values that do not refer to any of the variables used in the considered property, since these records do not affect its satisfaction; (b) removed, from each of the remaining records, the values of the variables that were not used in the property, while preserving the rest of the record; (c) generated missing values for the remaining variables by using various interpolation functions [33]. We considered different interpolation functions depending on the type of the signal, as commonly done in the literature (e.g., [2]).

Then, we analyzed each of the resulting $21\,952$ trace-property combinations. Since *TD-SB-TemPsy* aims to support engineers in detecting the source of property violations, we are interested in selecting the trace-property combinations that lead to such violations. We executed SB-TemPsy-Check by setting a timeout of $1\,\mathrm{min}$, thus enabling us to consider all the trace-property combinations in approximately $1\,\mathrm{min} \times 21952$=15 days of computation.

Out of the $21\,952$ trace-property combinations, 2328 of them timed out ($\approx 10.60\%$). The main reason behind such a timeout is the known scalability issue of the trace checking tool [1], especially when the property to check is defined with an order relationship pattern or an event scope.

Among the remaining $19\,624$ ($\approx 89.40\%$) trace-property combinations that did not timeout, $14\,940$ combinations represent traces that violate a property (76.13%). Though this may appear surprising at first, some of the properties only refer to specific phases of the satellite life cycle (e.g., satellite launch, deployment). We nevertheless checked these properties by considering all the traces provided by our industrial partner, including the ones that refer to the actual regular operations of the satellite. These trace-property combinations naturally led to a property violation[5].

Our final dataset contains $14\,940$ trace-property combinations leading to a property violation. In this dataset, the number of records in the traces ranges from 1 to $11\,901$ ($avg \approx 1032$, $StdDev \approx 1471$), the recording interval ranges from $0\,\mathrm{s}$, for traces with a single record, to $23\,\mathrm{h}.28\,\mathrm{min}$ ($avg = 4\,\mathrm{h}.51\,\mathrm{min}$, $StdDev = 6\,\mathrm{h}.24\,\mathrm{min}$). Notice that the number of records and the recording intervals of the traces of the final dataset are significantly smaller than those observed for the original traces. This is due to the fact that, for each trace-property combination, we removed from the trace all the records that only contained variable values that did not refer to any of the variables of the considered property (see step (a) above).

AFC dataset

The AFC benchmark model [32] used in the ARCH competition [19] comes with three properties (namely, AFC27, AFC29, and AFC33). We excluded property AFC27 because it could not be expressed with SB-TemPsy-DSL, since the language does not support nested operators[6]. Properties AFC29 and AFC33 have the same formula structure and differ only in terms of their parameters. We considered only one of them (property AFC29) and expressed it in SB-TemPsy-DSL. The property states that *"between 11 and 50 seconds, signal μ shall be lower than 0.007 "*, corresponding to the SB-TemPsy-DSL specification:

$$\phi_{29} \equiv \quad \texttt{\textbf{between}}\ 11\ \texttt{\textbf{and}}\ 50\ \texttt{\textbf{assert}}\ \mu\ < 0.007$$

As we are interested in reporting a diagnosis corresponding to the violation of AFC29, we used the ARIsTEO tool [34], a plugin for S-Taliro [35], to generate 10 traces that falsify the property, sampled over $50\,\mathrm{s}$ (i.e., a time horizon of $[0, 50]$), with simulations configured to use a variable sample step.

We therefore obtained 10 trace-property combinations; the number of records in the generated traces ranges from $22\,824$ to $23\,988$ ($avg = 23\,650$, $StdDev = 328$).

This is admittedly a much smaller dataset than the PROP-SAT one, if we consider the number of trace-property combinations it contains and the fact that we only considered one property. Moreover, given the simplicity of property AFC29, considering more traces would not lead to different diagnoses and conclusions (i.e., the catalogue of violation causes and corresponding diagnoses in *TD-SB-TemPsy* includes a single violation cause $\texttt{\textbf{c\_assert}}_1$ and one diagnosis $\texttt{\textbf{d\_assert}}_1$ for the **`assert`** construct supported by SB-TemPsy-DSL). We remark that, out of the eight STL properties included in the benchmark description [32], only three of them were included in the ARCH competition benchmark [19], and thus were known to be falsifiable. Nevertheless, this dataset is adequate for our goals, which are (a) to demonstrate that we can obtain similar results

[5] A property that does not refer to the regular operations of the satellite is expected to be violated if checked on a trace recording such regular operations.

[6] We refer the reader to our previous work [1] in which we discuss the expressiveness of SB-TemPsy-DSL

on a different dataset obtained from a publicly available benchmark model; (b) to support open science, using a non-proprietary dataset that can be made publicly available.

8.2 Applicability

We assessed the applicability of *TD-SB-TemPsy* by considering the 14940 trace-property combinations in the PROP-SAT dataset as well as the 10 trace-property combinations in the AFC one. Applicability entails the capacity to report diagnoses within reasonable time.

We remark that we could not perform a comparison between *TD-SB-TemPsy* and state-of-the-art tools, for a number of reasons. First, some alternative approaches [14, 15, 16] do not support signal-based temporal properties, thus making any comparison impossible. The only alternative that supports signal-based properties is AMT2.0 [13, 10]. However, the tool is no longer publicly available[7] and its successor *rtamt* [36] has dropped support for diagnostics capabilities, rendering impossible any experimental comparison.

8.2.1 PROP-SAT dataset

Methodology

We executed *TD-SB-TemPsy* on each trace-property combination in the PROP-SAT dataset, with a timeout of 1 min, leading to approximately 1 min × 14940=10 days of computation. For each execution, we recorded whether *TD-SB-TemPsy* finished within the timeout and the diagnoses (if any) it yielded. To assess the applicability of *TD-SB-TemPsy*, we analyzed the number of combinations in which *TD-SB-TemPsy* finished within the timeout and whether it yielded a diagnosis, i.e., whether at least one violation cause was applicable. We conducted our evaluation on a high-performance computing platform, using nodes equipped with Dell C6320 units (2 Xeon E5-2680v4@2.4 GHz, 128 GB).

Results

TD-SB-TemPsy finished within the timeout for ≈ 80.66% of the combinations (12 051 out of 14 940).

For the remaining 2889 combinations that *timed out*, ≈ 9.48% of these combinations (274 out of 2889) come from properties using the **`if-then`** construct, ≈ 76.71% of the combinations (2216 out of 2889) come from properties using the event scope constructs, and ≈ 13.81% of the combinations (399 out of 2889) come from properties that used the **`or`** and **`and`** operators to combine properties and clauses defined using the aforementioned constructs. For these combinations, the computational overhead to compute the diagnosis led to timeouts. Note that, in practice, engineers are likely to use larger timeouts than the one selected here, which is due to experimental constraints, and, therefore, we expect the percentage of combinations that time out to decrease.

For the 12 051 trace-property combinations that *finished within the timeout*, *TD-SB-TemPsy* always returned a diagnosis. We recall that a diagnosis is made by one or more diagnosis instances that are generated by *TD-SB-TemPsy* for the different atoms of the formula (see Section 4). These instances describe why the scope and pattern constructs of SB-TemPsy-DSL used for the definitions of the atom are violated by a trace. *TD-SB-TemPsy* produced one diagnosis instance for each atom of the formula (corresponding to the input property) for 94.38% of the combinations (11 374 out of 12 051). For the remaining 5.62% of the combinations (677 out of 12 051), some atoms of the formula did not lead to any diagnosis instance. In total, the 12 051 combinations returned 12 486 diagnosis instances.

Table 1
Number (#**N**) of diagnosis instances generated by *TD-SB-TemPsy* for each scope and pattern construct of SB-TemPsy-DSL (as used in the properties of our datasets).

Type	Construct	#N	Construct	#N	Construct	#N
			PROP-SAT dataset			
Scope	**`globally`**	0	**`before`**	294	**`after`**	2640
	`at`	1002	**`between`**	660		
Pattern	**`assert`**	7098	**`becomes`**	0	**`spike`**	321
	`oscillation`	460	**`rises`**	0	**`falls`**	0
	`overshoots`	0	**`undershoots`**	0	**`if-then`**	11
			MD1 dataset			
Pattern	**`becomes`**	656	**`overshoots`**	942	**`undershoots`**	940
			MD2 dataset			
Pattern	**`rises`**	90	**`falls`**	90		
			AFC dataset			
Pattern	**`assert`**	10				

The top part of Table 1 shows the number of diagnosis instances (column #**N**) computed by *TD-SB-TemPsy* for each scope and pattern construct of SB-TemPsy-DSL (as used in the properties of the PROP-SAT dataset). These results suggest that a relatively high percentage of the diagnosis instances (1002 + 294 + 660 + 2640 = 4596 out of 12 486, ≈ 36.81%) is related to scope constructs. Indeed, since we considered all the possible trace-property combinations in the dataset, there are many combinations for which the time instant values used to define the scope operators exceeded the maximum timestamp recorded in the trace. The remaining ≈ 63.19% of the diagnosis instances (7098 + 460 + 321 + 11 = 7890 out of 12 486) is related to patterns constructs. *TD-SB-TemPsy* returned diagnosis instances for the **`assert`**, **`spike`**, **`oscillation`**, and **`if-then`** constructs, though with different prevalence. *TD-SB-TemPsy* did not report any diagnosis instances for the **`becomes`**, **`rises`**, **`falls`**, **`overshoots`**, and **`undershoots`** constructs. We further analyzed the properties containing these constructs and noticed that, in all these cases, *TD-SB-TemPsy* detected a violation of the corresponding scope. In such cases, the diagnosis instances returned by *TD-SB-TemPsy* are only related to the scope constructs.

To guarantee a complete applicability assessment of *TD-SB-TemPsy*, covering all SB-TemPsy-DSL constructs, we built two additional datasets (MD1 and MD2), derived from the PROP-SAT one, using the following strategies:

MD1 (replacing the property scope). We considered all the 2562 trace-property combinations where the properties are defined using only one single pattern of type **`becomes`**, **`overshoots`**, or **`undershoots`** within a scope operator. We changed the scope of the patterns to **`globally`** in order to avoid any scope violations, thus making the detection of

[7] https://www-verimag.imag.fr/AMT-2-0.html

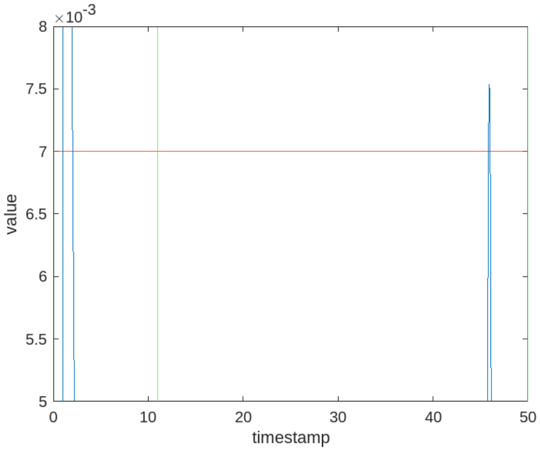


Figure 15. A trace (from the AFC dataset) with a signal violating the expression "**between** 11 **and** 50 **assert** μ < 0.007".

violations of these property patterns possible. As a result, we obtained 2562 additional trace-property combinations which constitute the MD1 dataset.

MD2 (changing the pattern definition). The patterns **`rises`** and **`falls`** were not used in any property containing only one single pattern, but were always used within the **`if-then`** construct. Therefore, we considered the 180 trace-property combinations where the properties contained the **`rises`** and **`falls`** patterns, and then we extracted from the **`if-then`** construct the subproperties that were using these patterns. This led to 180 additional trace-property combinations, which constitute the MD2 dataset.

We executed *TD-SB-TemPsy* on the MD1 and MD2 datasets with a timeout of $1\,\mathrm{min}$. In the case of MD1, *TD-SB-TemPsy* yielded diagnosis instances for $\approx$ 99.06% of the combinations (2538 out of 2562), with no timeout; the distribution of these instances is shown in the second block (from the top) of Table 1. For MD2, *TD-SB-TemPsy* yielded diagnosis instances for all 180 combinations; the distribution of these instances is shown in the third block (from the top) of Table 1.

8.2.2 AFC dataset

Methodology

We executed *TD-SB-TemPsy* on each of the 10 trace-property combinations in the AFC dataset, with a timeout of $1\,\mathrm{min}$. The total time to execute *TD-SB-TemPsy* on all these trace-property combinations was $\approx 14\,\mathrm{s}$; hence, no timeouts occurred.

Also in this case, we assessed the applicability of *TD-SB-TemPsy* by analyzing all the 10 trace-property combinations processed by *TD-SB-TemPsy*, checking whether it yielded a diagnosis.

Results

TD-SB-TemPsy yielded diagnosis instances for all the 10 trace-property combinations in the AFC dataset; all these diagnoses were of type **`d_assert`**$_1$.

For instance, the signal in Figure 15 violates property ϕ_{29}, showing at least one timestamp in which the signal violates the condition ($\mu < 0.007$). More specifically, the signal violates the condition for 99 records within the time interval $[45.85, 46.014]$, which lies within the scope interval $[11, 50]$ in ϕ_{29} (delimited by green vertical lines in the figure), with values ranging between 0.0070103 and 0.0075373. According to the definition of the diagnosis **`d_assert`**$_1$, *TD-SB-TemPsy* reports the first record (i.e., timestamp and the corresponding signal value) in which the condition was violated. The diagnosis is then as follows: $\langle 45.85, 0.0070278\rangle$. The choice of reporting the first record that violates the condition is motivated by the fact that we are interested in detecting and reporting the root cause(s) of the property violation (See section 6.1).

8.3 Discussion

The results show that *TD-SB-TemPsy* was widely applicable in the context of the two datasets we considered, including one based on an industrial case study.

Indeed, when considering the PROP-SAT dataset as well as MD1 and MD2, *TD-SB-TemPsy* was able to finish within the small timeout of $1\,\mathrm{min}$ for $12\,051 + 2562 + 180 = 14\,793$ out of $14\,940 + 2562 + 180 = 17\,682$ ($\approx$ 83.66%) of the trace-property combinations in this group of datasets, returning a diagnosis for $12\,051 + 2538 + 180 = 14\,769$ combinations ($\approx$ 99.84% of the cases). Moreover, in the case of the AFC dataset, *TD-SB-TemPsy* processed all the 10 trace-property combinations well below the timeout, yielding a diagnosis for all of them.

These results suggest that, in practice, our set of violation causes provide sufficient coverage of observed violations.

8.3.1 Threats to Validity

In terms of *internal validity*, the choice of a timeout of $1\,\mathrm{min}$, justified by the high computational time (15 days) for the experiments on the PROP-SAT dataset, led to a number of trace-property combinations for which *TD-SB-TemPsy* did not finish its execution. Considering a larger timeout would further increase the applicability of *TD-SB-TemPsy*. Moreover, we assumed that the traces in the PROP-SAT dataset (provided by our industrial partner) were correctly collected after the satellite deployment. The possible presence of erroneous records might lead to a different number of trace-property combinations leading to a violation, and to different diagnosis instances. Furthermore, we have assumed that the verdicts reported by SB-TemPsy-Check were correct.

In terms of *external validity*, the trace-property combinations in the PROP-SAT dataset may be a threat for the generalization of our results, as other datasets may differ in terms of (a) the constructs used for expressing the properties, (b) the type of property violations. We mitigated this threat by selecting an industrial case study in the satellite domain that is representative of complex CPS, with large traces and many complex properties elicited with experts. Further, we modified scopes and patterns in the properties of the PROP-SAT dataset to expand our analysis such as to consider more trace-property combinations. Moreover, we also considered an additional dataset (AFC) from the benchmark used for a popular competition for the falsification of temporal logic specifications over CPS.

Regarding *conclusion validity*, the trace-property combinations in our datasets did not trigger all the 34 violation

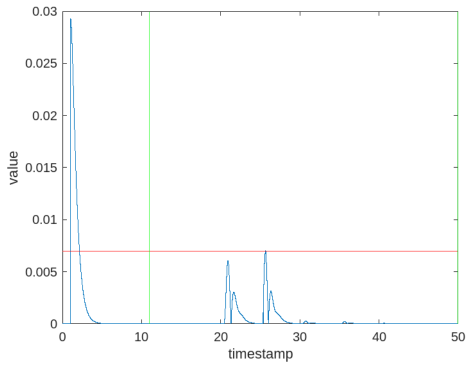


Figure 16. A trace (from the AFC dataset) with a signal violating the expression "**between** 11 **and** 50 **assert** mu < 0.007" (with no zoom factor in the plot).

causes in our catalogue (see Figure 7). More in details, the trace-property combinations in the PROP-SAT and AFC datasets cover 9 out of the 34 violation causes. If we include the two additional datasets MD1 and MD2, the total number of covered violation causes reach 17 out of 34 (8 new ones). These 8 new covered violations causes are distributed as follows: 2 **becomes**, 2 **rises** (and its dual **falls**) and 4 **overshoots** (and its dual **undershoots**).

8.4 Data Availability

We cannot publicly release the traces and properties used in the experiments for the PROP-SAT because they are subject to a non-disclosure agreement. We make the raw output of *TD-SB-TemPsy*, the traces generated as part of the AFC dataset, and the script used for the analysis of the evaluation data available as supplementary material in a permanent repository [37]. *TD-SB-TemPsy* is available under the Apache 2.0 license at https://github.com/SNTSVV/TD-SB-TemPsy; a permanent record is also available on Figshare [38].

9 Practical Implications

9.1 Usefulness of the diagnoses

When engineers use a run-time verification tool that only yields Boolean verdicts, if a property is violated on a trace, engineers have to inspect the trace to understand the cause of the violation.

Such an inspection is not necessarily trivial, especially for the more complex types of properties (e.g., those involving spike or oscillatory behaviors), and cannot rely on a simple visualization of the signals. This problem is even more noticeable when dealing with huge execution traces, containing thousands of records.

For example, let us consider property ϕ_{29} from the AFC dataset. Figure 16 shows one of the execution traces considered in our evaluation. Given the small order of magnitude used in the numeric parameters of the property (e.g., the threshold 0.007) and the shape of the signal, detecting the violation through a visual inspection is not immediate, even if a large zoom factor is used in the visualization.

On the other hand, if an engineer uses our approach, she can immediately look up the important record in the trace, since it is indicated in the diagnosis $\langle 45.85, 0.0070278\rangle$.

Note that the example above is based on a simple assertion property. Automating the diagnostics of violation is even more important for more complex types of properties (e.g., those involving spike or oscillatory behaviors).

The actual effort savings are specific to individual case studies and can only be estimated through a user study. We plan to conduct one as part of future work.

9.2 Extending the catalogue of violation patterns and diagnoses

As discussed in section 1, our catalogue of 34 violation causes, each associated with one diagnosis, *is not complete*. Nevertheless, following the methodology illustrated in section 5, users can add new violation causes depending on their specific needs or on the requirements of particular domains.

We remark that this extension of the catalogue is a one-time effort. It can be performed by engineers following the three steps: behavior analysis, definition of violation causes, and definition of diagnoses. In particular, step "Definition of violation causes" requires new violation causes to be checked (for correctness) by verifying whether a formula (obtained from the formal specification of the violation cause semantics) is unsatisfiable (see section 5.4). This check can be performed with state-of-the-art constraint solvers like Z3. Overall, fulfilling this requirement prevents the users from introducing errors in the definition of violation causes.

Finally, even if the original catalogue of 34 violation causes is not complete, we remark that it results from an extended industrial case study and relies on a taxonomy [9] of pattern-based constructs that have been identified through a thorough review of the literature, whose completeness has been validated in an industrial context. Based on this, we expect our catalogue to be widely reusable in different CPS domains, provided that the requirements to be checked using a run-time verification tool can be expressed using SB-TemPsy-DSL.

10 Related Work

The problem of enriching Boolean verification verdicts with additional information that supports reasoning on the causes of such verdicts has been widely studied in the literature. This section discusses related work in the trace-checking and model-checking areas. We included the latter since a trace can be seen as a model made by a sequence of consecutive states, each representing one trace record, with transitions connecting the consecutive records.

In the *trace-checking* area, there are two main strategies (see Section 1) that aim to provide additional information on the causes of a property violation: (i) isolating slices of the traces that explain the property violation (e.g., [10, 11, 12, 13]); and (ii) checking whether the traces show common behaviors that lead to the property violation (e.g., [15, 16, 14]). The first strategy produces large explanations for complex properties since the size of the explanation increases with the number of operators of the formula expressing the property of interest. Existing approaches based on the second strategy do not support complex signal-based temporal properties (as the ones considered in this work)

Table 2
Comparison of trace diagnostic approaches

Approach	SBTP	Lang.	Method.	Eval.
Ferrère et al. [10]	+	TL	-	T
Mukherjee and Dasgupta [11]	-	TL	-	P
Beer et al. [12]	-	TL	-	P
Ničković et al. [13]	+	TL	-	P
Dawes and Reger [14]	-	TL	-	P
Dou et al. [15]	-	DSL	-	S
Luo et al. [16]	-	DSL	-	P
TD-SB-TemPsy	+	DSL	+	I,P

and are not complemented by a precise methodology that describes how to add new causes that support more complex properties. Therefore, in this work we have proposed a novel, language-agnostic methodology for defining *violation causes* and *diagnoses*, and applied it in the context of signal-based temporal properties expressed in SB-TemPsy-DSL.

Table 2 provides a comparison, in terms of trace diagnostics support, of the aforementioned trace checking approaches. Column *SBTP* indicates whether the approach supports signal-based temporal properties. Column *Lang.* indicates — using the symbols DSL and TL — whether the approach supports, respectively, a high-level, DSL-like specification language or a low-level, temporal-logic language. Column *Method.* indicates whether the diagnostic approach supports a methodology to add new violation causes. Column *Eval.* indicates the type(s) of benchmarks used in the evaluation of the approach (*I*: industrial case study, *P* public benchmark, *S*: synthetic benchmark, *T*: toy example).

As shown in the table, *TD-SB-TemPsy* is the only approach that supports signal-based temporal properties expressed in a DSL-like specification language, that is complemented by a methodology allowing users to define new violation causes, and that has been evaluated using datasets derived both from a complex industrial case study and from a public benchmark.

In the *model-checking* area, some approaches (e.g., [39, 40, 41, 42, 43, 44, 45, 46, 47, 48]) extract information from the model (e.g., model slices) to explain the model checking verdict. Typically, these approaches have limited scalability and therefore are not easily applicable to the trace-checking scenario. *TD-SB-TemPsy* relies on a conceptually different technique, which leverages violation causes and diagnoses to explain trace checking verdicts. Moreover, its implementation uses existing technologies that showed encouraging scalability results in previous works; our evaluation confirms the applicability of our solution. Other approaches (e.g., [49, 50, 51, 52, 53, 54, 55]) rely on deductive reasoning techniques to explain model checking verdicts. Different from the approach proposed in this work, they usually provide an exhaustive explanation for a verdict by considering some initial assertions (e.g., simple conditions on the values assumed by the variables in the states of the model) and examining how logical operators can be applied to reach a specific logical conclusion. However, the proofs produced by deductive reasoning approaches are usually difficult to understand for non-experts. Besides, their size significantly grows with the size of the model to analyze [56]. Therefore, when the model represents a trace, which is typically large in practice (i.e., because of a large number of records), the generated proofs are likely to be extremely large and difficult to understand by engineers.

11 Conclusion

In this paper, we proposed *TD-SB-TemPsy*, a trace-diagnostic approach for signal-based temporal properties, based on violation causes and diagnoses. We defined a methodology for defining violation causes and diagnoses that provides formal soundness guarantees. We proposed a catalog of 34 violation causes, each associated with one diagnosis, for properties expressed in SB-TemPsy-DSL. We evaluated *TD-SB-TemPsy* by assessing its applicability on on two datasets, including one based on a complex industrial case study.

For the latter, *TD-SB-TemPsy* finished within the stringent timeout of $1\,\mathrm{min}$ for $\approx 83.66\%$ of the trace-property combinations, yielding a diagnosis in $\approx 99.84\%$ of these cases; moreover, it also yielded a diagnosis, within the same timeout, for all the trace-property combinations in the other dataset. In practice, outside of experimental settings, longer timeouts can be considered.

In the future, we plan to perform a large-scale, systematic evaluation to assess (a) the scalability of *TD-SB-TemPsy* with respect to the trace size and (b) its applicability when dealing with different violation causes. This evaluation requires the use of synthesized traces, which enable varying the trace size and controlling the causes of property violations. Furthermore, we are going to assess the applicability of *TD-SB-TemPsy* on a diverse set of CPS case studies (e.g., unmanned aerial vehicles). Moreover, we expect to revisit the implementation of *TD-SB-TemPsy* and SB-TemPsy-Check, to support a tighter integration between the two tools, so that some intermediate outputs for the trace diagnostics procedure could be computed during the execution of the trace checking one. In addition, we intend to conduct a user study to assess the usefulness of the diagnoses provided by *TD-SB-TemPsy*, for example in the context of fault localization.

Acknowledgments

Part of this work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), through its DG and CRC programs; by European Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 957254 (COSMOS).

References

[1] C. Boufaied, C. Menghi, D. Bianculli, L. Briand, and Y. Isasi Parache, "Trace-checking signal-based temporal properties: A model-driven approach," in *International Conference on Automated Software Engineering (ASE)*. New York, NY, USA: IEEE/ACM, 2020, pp. 1004–1015.

[2] C. Menghi, E. Viganò, D. Bianculli, and L. C. Briand, "Trace-Checking CPS Properties: Bridging the Cyber-Physical Gap," in *International Conference on Software Engineering (ICSE)*. Los Alamitos, CA, USA: IEEE, 2021, pp. 847–859.

[3] C. Menghi, S. Nejati, K. Gaaloul, and L. C. Briand, "Generating automated and online test oracles for simulink models with continuous and uncertain behaviors," in *European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. New York, NY, USA: ACM, 2019, pp. 27–38.

[4] F. Gorostiaga and C. Sánchez, "Striver: Stream runtime verification for real-time event-streams," in *International Conference on Runtime Verification (RV)*. Cham: Springer, 2018, pp. 282–298.

[5] L. Convent, S. Hungerecker, M. Leucker, T. Scheffel, M. Schmitz, and D. Thoma, "TeSSLa: temporal stream-based specification language," in *Brazilian Symposium on Formal Methods*. Cham: Springer, 2018, pp. 144–162.

[6] P. Faymonville, B. Finkbeiner, S. Schirmer, and H. Torfah, "A stream-based specification language for network monitoring," in *International Conference on Runtime Verification*, vol. 10012, 09 2016. [Online]. Available: https://10.1007/978-3-319-46982-9_10

[7] C. Menghi, C. Tsigkanos, P. Pelliccione, C. Ghezzi, and T. Berger, "Specification patterns for robotic missions," *IEEE Transactions on Software Engineering*, vol. 47, no. 10, pp. 2208–2224, 2021.

[8] M. Fowler, *Domain-specific languages*. Boston, MA, USA: Pearson Education, 2010.

[9] C. Boufaied, M. Jukss, D. Bianculli, L. C. Briand, and Y. Isasi Parache, "Signal-based properties of cyber-physical systems: Taxonomy and logic-based characterization," *Journal of Systems and Software*, vol. 174, p. 110881, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121220302715

[10] T. Ferrère, O. Maler, and D. Ničković, "Trace diagnostics using temporal implicants," in *International Symposium on Automated Technology for Verification and Analysis*. Cham: Springer, 2015, pp. 241–258.

[11] S. Mukherjee and P. Dasgupta, "Computing minimal debugging windows in failure traces of ams assertions," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 11, pp. 1776–1781, 2012.

[12] I. Beer, S. Ben-David, H. Chockler, A. Orni, and R. Trefler, "Explaining counterexamples using causality," in *International Conference on Computer Aided Verification (CAV)*. Berlin, Heidelberg: Springer, 2009, pp. 94–108.

[13] D. Ničković, O. Lebeltel, O. Maler, T. Ferrère, and D. Ulus, "Amt 2.0: Qualitative and quantitative trace analysis with extended signal temporal logic," in *Tools and Algorithms for the Construction and Analysis of Systems*. Cham: Springer, 2018, pp. 303–319.

[14] J. H. Dawes and G. Reger, "Explaining violations of properties in control-flow temporal logic," in *International Conference on Runtime Verification (RV)*. Cham: Springer, 2019, pp. 202–220.

[15] W. Dou, D. Bianculli, and L. Briand, "Model-driven trace diagnostics for pattern-based temporal specifications," in *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. New York, NY, USA: ACM, 2018, pp. 278–288.

[16] Q. Luo, Y. Zhang, C. Lee, D. Jin, P. O. Meredith, T. Serbanuta, and G. Rosu, "Rv-monitor: Efficient parametric runtime verification with simultaneous properties," in *Runtime Verification (RV 2014)*, ser. Lecture Notes in Computer Science, vol. 8734. Cham: Springer, 2014, pp. 285–300. [Online]. Available: https://doi.org/10.1007/978-3-319-11164-3_24

[17] W. Dou, D. Bianculli, and L. Briand, "A model-driven approach to trace checking of pattern-based temporal properties," in *Proc. MODELS2017*. Los Alamitos, CA, USA: IEEE Computer Society, 2017, pp. 323–333.

[18] D. Giannakopoulou, T. Pressburger, A. Mavridou, and J. Schumann, "Generation of formal requirements from structured natural language," in *Requirements Engineering: Foundation for Software Quality (REFSQ 2020)*. Cham: Springer International Publishing, 2020, pp. 19–35.

[19] G. Ernst, P. Arcaini, I. Bennani, A. Chandratre, A. Donzé, G. Fainekos, G. Frehse, K. Gaaloul, J. Inoue, T. Khandait *et al.*, "Arch-comp 2021 category report: Falsification with validation of results." in *ARCH@ ADHS*, 2021, pp. 133–152.

[20] Wikipedia contributors, "Beta angle — Wikipedia, the free encyclopedia," 2021, [Online; accessed 2-September-2021]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Beta_angle&oldid=1035086598

[21] T. Ott, A. Benoit, P. Van den Braembussche, and W. Fichter, "ESA pointing error engineering handbook," in *8th International ESA Conference on Guidance, Navigation & Control Systems*. Bruxelles: European Space Agency, 2011, p. 17.

[22] A. J. Robinson and A. Voronkov, *Handbook of automated reasoning*. Amsterdam, Holland: Elsevier, 2001, vol. 1.

[23] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 337–340.

[24] D. Giannakopoulou, T. Pressburger, A. Mavridou, and J. Schumann, "Automated formalization of structured natural language requirements," *Information and Software Technology*, vol. 137, p. 106590, 2021.

[25] OMG, "ISO/IEC 19507 (OCL v2.3.1)," http://www.omg.org/spec/OCL/ISO/19507/PDF, April 2012.

[26] E. Bartocci, Y. Falcone, B. Bonakdarpour, C. Colombo, N. Decker, K. Havelund, Y. Joshi, F. Klaedtke, R. Milewicz, G. Reger, G. Rosu, J. Signoles, D. Thoma, E. Zalinescu, and Y. Zhang, "First international competition on runtime verification: rules, benchmarks, tools, and final results of CRV 2014," *Int. J. Softw. Tools Technol. Transf.*, vol. 21, no. 1, pp. 31–70, 2019. [Online]. Available: https://doi.org/10.1007/s10009-017-0454-5

[27] G. Reger, "A report of rv-cubes 2017," in *RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools, September 15, 2017, Seattle, WA, USA*, ser. Kalpa Publications in Computing, G. Reger and K. Havelund, Eds., vol. 3. EasyChair, 2017, pp. 1–9. [Online]. Available: https://doi.org/10.

29007/2496

[28] S. Krstić and J. Schneider, "A benchmark generator for online first-order monitoring," in *International Conference on Runtime Verification*. Springer, 2020, pp. 482–494.

[29] J. Li and K. Y. Rozier, "Mltl benchmark generation via formula progression," in *International Conference on Runtime Verification*. Springer, 2018, pp. 426–433.

[30] D. Ulus, "Timescales: A benchmark generator for mtl monitoring tools," in *International Conference on Runtime Verification*. Springer, 2019, pp. 402–412.

[31] C. Menghi, E. Viganò, D. Bianculli, and L. C. Briand, "Trace-checking cps properties: Bridging the cyber-physical gap," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 847–859.

[32] X. Jin, J. V. Deshmukh, J. Kapinski, K. Ueda, and K. Butts, "Powertrain control verification benchmark," in *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 253–262. [Online]. Available: https://doi.org/10.1145/2562059.2562140

[33] J. Szabados and P. Vértesi, *Interpolation of functions*. Singapore: World Scientific, 1990.

[34] C. Menghi, S. Nejati, L. C. Briand, and P. Yago Isasi, "Approximation-refinement testing of compute-intensive cyber-physical models: An approach based on system identification," in *Proc. ICSE 2020*. New York, NY, USA: ACM, 2020.

[35] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-taliro: A tool for temporal logic falsification for hybrid systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer, 2011, pp. 254–257.

[36] D. Ničković and T. Yamaguchi, "Rtamt: Online robustness monitors from stl," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2020, pp. 564–571.

[37] C. Boufaied, "TD-SB-TemPsy Supplementary Material," 1 2023. [Online]. Available: https://figshare.com/articles/software/TD-SB-TemPsy_SupplementaryMaterial/21956924

[38] C. Boufaied, C. Menghi, D. Bianculli, and L. Briand, "TD-SB-TemPsy software artifact," 1 2023. [Online]. Available: https://figshare.com/articles/software/TD-SB-TemPsy/21954563

[39] C. Menghi, A. M. Rizzi, and A. Bernasconi, "Integrating topological proofs with model checking to instrument iterative design," in *Fundamental Approaches to Software Engineering (FASE)*. Cham: Springer, 2020, pp. 53–74.

[40] V. Schuppan, "Towards a notion of unsatisfiable and unrealizable cores for ltl," *Science of Computer Programming*, vol. 77, no. 7-8, pp. 908–939, 2012.

[41] F. Hantry and M.-S. Hacid, "Handling conflicts in depth-first search for ltl tableau to debug compliance based languages," *Electronic Proceedings in Theoretical Computer Science*, vol. 68, 09 2011.

[42] G. Zheng, T. Nguyen, S. G. Brida, G. Regis, M. F. Frias, N. Aguirre, and H. Bagheri, "Flack: Counterexample-guided fault localization for alloy models," in *International Conference on Software Engineering (ICSE)*. Los Alamitos, CA, USA: IEEE, 2021, pp. 637–648.

[43] M. Chechik and A. Gurfinkel, "A framework for counterexample generation and exploration," in *Fundamental Approaches to Software Engineering*, ser. FASE. Berlin, Heidelberg: Springer, 2005, p. 220–236.

[44] T. Bochot, P. Virelizier, H. Waeselynck, and V. Wiels, "Paths to property violation: A structural approach for analyzing counter-examples," in *International Symposium on High Assurance Systems Engineering*. Los Alamitos, CA, USA: IEEE, 2010, pp. 74–83.

[45] A. Griggio, M. Roveri, and S. Tonetta, "Certifying proofs for ltl model checking," in *Formal Methods in Computer Aided Design (FMCAD)*. Los Alamitos, CA, USA: IEEE, 2018, pp. 1–9.

[46] F. Funke, S. Jantsch, and C. Baier, "Farkas certificates and minimal witnesses for probabilistic reachability constraints," in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2020)*, ser. LNCS, vol. 12078. Cham: Springer, 2020, pp. 324–345.

[47] N. Timm, S. Gruner, M. Nxumalo, and J. Botha, "Model checking safety and liveness via k-induction and witness refinement with constraint generation," *Science of Computer Programming*, vol. 200, p. 102532, 2020.

[48] A. Gurfinkel and M. Chechik, "Proof-like counter-examples," in *Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer, 2003, pp. 160–175.

[49] D. Peled and L. Zuck, "From model checking to a temporal proof," in *International SPIN workshop on Model checking of software*. Berlin Heidelberg: Springer-Verlag, 2001, pp. 1–14.

[50] A. Bernasconi, C. Menghi, P. Spoletini, L. D. Zuck, and C. Ghezzi, "From model checking to a temporal proof for partial models," in *Software Engineering and Formal Methods (SEFM)*. Cham: Springer, 2017, pp. 54–69.

[51] D. Peled, A. Pnueli, and L. Zuck, "From falsification to verification," in *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Berlin, Heidelberg: Springer, 2001, pp. 292–304.

[52] A. Mebsout and C. Tinelli, "Proof certificates for smt-based model checkers for infinite-state systems," in *2016 Formal Methods in Computer-Aided Design (FMCAD)*. Los Alamitos, CA, USA: IEEE, 2016, pp. 117–124.

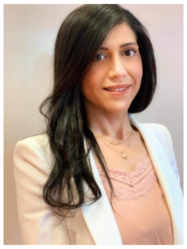
[53] D. Basin, B. N. Bhatt, and D. Traytel, "Optimal proofs for linear temporal logic on lasso words," in *International Symposium on Automated Technology for Verification and Analysis*. Cham: Springer, 2018, pp. 37–55.

[54] A. Pnueli and Y. Kesten, "A deductive proof system for ctl," in *International Conference on Concurrency Theory*. Berlin, Heidelberg: Springer-Verlag, 2002, pp. 24–40.

[55] I. Balaban, A. Pnueli, and L. D. Zuck, "Proving the refuted: Symbolic model checkers as proof generators," in *Concurrency, Compositionality, and Correctness*. Berlin, Heidelberg: Springer, 2010, pp. 221–236.

[56] S. Grebing and M. Ulbrich, "Usability recommendations for user guidance in deductive program verification," in *Deductive Software Verification: Future Perspectives: Reflections on the Occasion of 20 Years of KeY*.

Cham: Springer, 2020, pp. 261–284. [Online]. Available: https://doi.org/10.1007/978-3-030-64354-6_11



Chaima Boufaied is a postdoctoral fellow at the University of Ottawa. She got a PhD degree in Informatics from the University of Luxembourg in April 2021; between February 2021 and January 2022 she was a Research Associate at the Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg. Chaima's research focuses on the specification and verification of cyber-physical systems, with particular interest in run-time verification and log analysis.

Claudio Menghi is an Assistant Professor at the University of Bergamo. After receiving his Ph.D. at Politecnico di Milano, he was a postdoctoral researcher at Chalmers | University of Gothenburg (Sweden), a Research Associate at the University of Luxembourg (Luxembourg), and an Assistant Professor at McMaster University (Canada). His current research interests lie in software engineering, focusing on cyber-physical systems and formal verification.

Domenico Bianculli is associate professor/chief scientist 2 at the Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg. He holds a PhD degree from Università della Svizzera italiana (Lugano, Switzerland), a MSc in Computing Systems Engineering and a BSc in Computer Engineering, both from Politecnico di Milano (Milan, Italy). Domenico's research focuses on the specification and verification of evolvable software systems. His research interests include: run-time verification, log analysis, program analysis, and access control.

Lionel C. Briand is professor of software engineering and has shared appointments between (1) School of Electrical Engineering and Computer Science, University of Ottawa, Canada and (2) The SnT centre for Security, Reliability, and Trust, University of Luxembourg. He is the head of the SVV department at the SnT Centre and a Canada Research Chair in Intelligent Software Dependability and Compliance (Tier 1).

He holds an ERC Advanced Grant, the most prestigious European individual research award, and has conducted applied research in collaboration with industry for more than 25 years, including projects in the automotive, aerospace, manufacturing, financial, and energy domains. He was elevated to the grades of IEEE and ACM fellow, granted the IEEE Computer Society Harlan Mills award (2012) and the IEEE Reliability Society Engineer-of-the-year award (2013) for his work on software verification and testing. His research interests include: Model-driven development, testing and verification, search-based software engineering, requirements engineering, and empirical software engineering.