



HAL
open science

Mining of extended signal temporal logic specifications with ParetoLib 2.0

Akshay Mambakam, José Ignacio Requeno Jarabo, Alexey Bakhirkin, Nicolas Basset, Thao Dang

► **To cite this version:**

Akshay Mambakam, José Ignacio Requeno Jarabo, Alexey Bakhirkin, Nicolas Basset, Thao Dang. Mining of extended signal temporal logic specifications with ParetoLib 2.0. *Formal Methods in System Design*, 2024, 62 (1-3), pp.260-284. 10.1007/s10703-024-00453-2 . hal-04794812

HAL Id: hal-04794812

<https://hal.science/hal-04794812v1>

Submitted on 21 Nov 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.


L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License



Mining of extended signal temporal logic specifications with ParetoLib 2.0

Akshay Mambakam¹ · José Ignacio Requeno Jarabo²  · Alexey Bakhirkin^{1,2} · Nicolas Basset¹ · Thao Dang¹

Received: 10 January 2023 / Accepted: 14 April 2024 / Published online: 6 May 2024
© The Author(s) 2024

Abstract

Cyber-physical systems are complex environments that combine physical devices (i.e., sensors and actuators) with a software controller. The ubiquity of these systems and dangers associated with their failure require the implementation of mechanisms to monitor, verify and guarantee their correct behaviour. This paper presents ParetoLib 2.0, a Python tool for offline monitoring and specification mining of cyber-physical systems. ParetoLib 2.0 uses signal temporal logic (STL) as the formalism for specifying properties on time series. ParetoLib 2.0 builds upon other tools for evaluating and mining STL expressions, and extends them with new functionalities. ParetoLib 2.0 implements a set of new quantitative operators for trace analysis in STL, a novel mining algorithm and an original graphical user interface. Additionally, the performance is optimised with respect to previous releases of the tool via data-type annotations and multi core support. ParetoLib 2.0 allows the offline verification of STL properties as well as the specification mining of parametric STL templates. Thanks to the implementation of the new quantitative operators for STL, the tool outperforms the expressiveness and capabilities of similar runtime monitors.

Keywords Signal Temporal Logic · Quantitative analysis · Specification mining · Parameter synthesis · Python

1 Introduction

Cyber-physical systems are complex environments that combine physical devices (i.e., sensors and actuators) with a software controller. The ubiquity of these systems and dangers associated with their failure require the implementation of mechanisms to monitor, verify and guarantee their correct behaviour.

This paper presents ParetoLib 2.0, a Python tool for offline monitoring of cyber-physical systems. ParetoLib 2.0 allows qualitative and quantitative analysis of real-time signals using specification in an extension of Signal Temporal Logic (STL) [1]. Additionally, ParetoLib 2.0 supports mining parameter valuations of parametric STL (PSTL) [2] properties

Akshay Mambakam, José Ignacio Requeno Jarabo, Alexey Bakhirkin, Nicolas Basset and Thao Dang have contributed equally to this work.

Extended author information available on the last page of the article

from trace examples (i.e., a variant of STL that allows parameters in addition to numeric constants).

ParetoLib 2.0 gathers the contributions presented in [3–5] and extends previous versions of ParetoLib [6] with the following additions. First, we provide a graphical user interface that helps the end users efficiently interact with the tools and interpret the results. Second, we support additional quantitative STL operators allowing to express a richer set of properties involving *counting of events* (*e-count*), *trends* (derivatives), or *accumulations* (integrals). Third, ParetoLib includes recent algorithms for mining parametric STL specifications that do not require monotonicity assumptions of the validity domain [7]. Finally, we have optimised the tool performance by adding multicore support, type annotations and directives to compile the Python modules into C code.

The paper is organised as follows: Sect. 2 motivates the extension of STL with some examples that are complicated to express without the new STL operators. Section 3 introduces the STL specification language and the quantitative extensions. Next, Sect. 4 presents the technical aspects of ParetoLib 2.0 and details the new contributions. Section 5 presents a case study that uses the new quantitative operators. Section 6 compares ParetoLib 2.0 to similar tools. Finally, Sect. 7 gathers the conclusions and sketches the future work.

2 Motivating examples

Before formally defining the STL language, let us look at some examples of properties that we would like to express. In particular, we look at properties that motivated the development of more expressive and harder to monitor logics.

Example 1 (Stabilisation) The first interesting property is stabilisation around a value that is not known in advance, e.g., “ x stays within $+/-0.05$ units of some value for at least 200 time units”. It is tempting to formalise this property using existential quantification “there exists a threshold v , such that...”, which is possible with first-order logic of signals (and was one of its motivational properties [8]), but it is actually not necessary. Instead, we can compute the minimum and maximum of x over the next 200 time units and compare their distance to $0.1 = 2 \cdot 0.05$. In some imaginary language, we could write $\max_{[0,200]}x - \min_{[0,200]}x \leq 0.1$. At this point we propose to separate the aggregate operators from the operator that defines the temporal window, which will be useful later, when the *until* operator will define a window of variable width. We use the operator $On_{[a,b]}$ to define the temporal window of constant width and the operators *Min* and *Max* (capitalised) to denote the minimum and maximum over the previously defined window.

Signal x stabilises within 0.05 units of an unknown value for 200 time units:

$$On_{[0,200]}Max x - On_{[0,200]}Min x \leq 0.1$$

Figure 1 shows an example of a signal $x(t)$ (red) performing damped oscillation with the period of 250 time units. Blue and green curves are the maximum and the minimum of x over a sliding window $[t, t + 200]$. Finally, the orange Boolean signal (its y scale is on the right) evaluates to true (i.e., $y = 1$) when the maximum and minimum of x over the next 200 time units are within 0.1.

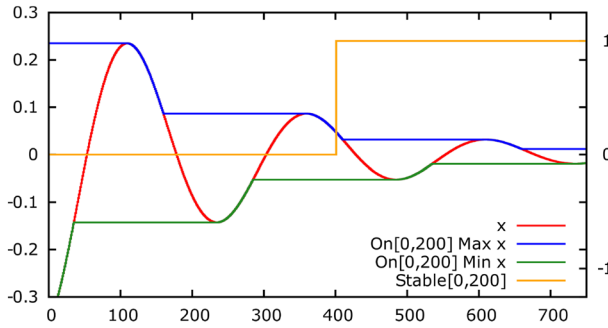
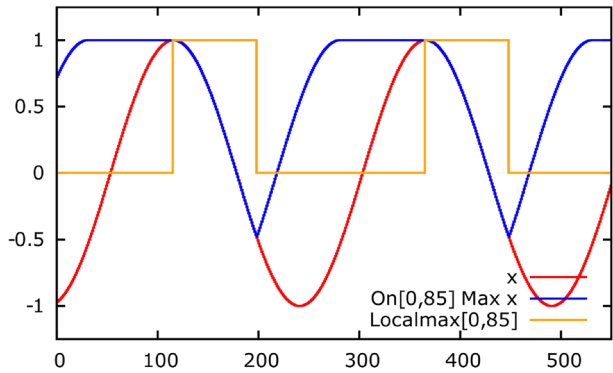


Fig. 1 Damped oscillation $x(t)$ and its maximum and minimum over the window $[t, t + 200]$

Fig. 2 Sine wave $x(t)$, its maximum over the window $[t, t + 200]$, and whether $x(t)$ is a local maximum on the interval $[t, t + 200]$



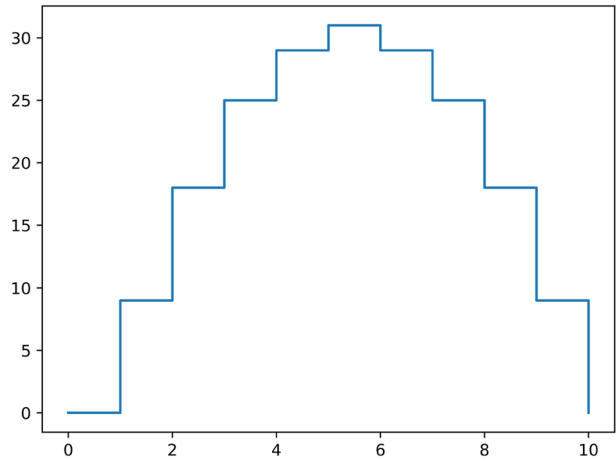
Example 2 (Local Maximum) Consider the property: “the current value of x is a minimum or maximum in some neighbourhood of current time point”. Previously, a similar property became a motivation to extend STL with freeze quantifiers [9], but we can also express it by comparing the value of a signal with some aggregate information about its neighbourhood, which we can do similarly to the previous example.

Current value of x is a local maximum on the interval $[0, 85]$ relative to the current time.

$$x \geq On_{[0,85]}Max x$$

Figure 2 shows an example of a sine wave $x(t)$ (red) with the period of 250 time units. Blue curve is the maximum x over a sliding window $[t, t + 85]$. The orange Boolean signal evaluates to true when the current value of x is a maximum for the next 85 time units. Another way of detecting the moments when the signal reaches a local minimum or maximum, regardless of the actual value of x , consists in expressing the property in terms of derivatives (D operator). Local maximum and minimum are reached when derivatives are equal to zero, so the following expression will draw a pulse in the surroundings of local peaks in the orange Boolean signal every 125 time units (not shown in Fig. 2).

Fig. 3 Piece-wise constant signal $w_{10}(t)$



$$On_{[0,85]}|Dx| \leq 0.1$$

Example 3 (Stabilisation Contd.) We want to be able to assert that x becomes stable around some value not for a fixed duration, but until some signal q becomes true. We will be able to do this with our version of *until* operator.

Signal x is stable within 0.05 units of an unknown value until q becomes true:

$$(Max\ x\mathcal{U}q) - (Min\ x\mathcal{U}q) \leq 0.1$$

Intuitively, for a given time point, we want the monitor to find the closest future time point, where q holds and compute *Min* and *Max* of x over the resulting interval. Note that this property cannot be easily monitored in the framework of “STL with pre-processing”, since it requires the monitor to compute *Min* and *Max* over a sliding window of variable width, which depends on the satisfaction signal of q .

Example 4 (Linear Increase) At this point, we can assert x to follow a more complex shape, for example, to increase or decrease with a given slope. Let T denote an auxiliary signal that linearly increases with rate 1 (like a clock of a timed automaton), i.e. we define $T(t) = t$; this example works as well for $T(t) = t + c$, where c is a constant. To specify that x increases with the rate 2.5, we assert that the distance from x to $2.5 \cdot T$ stays within some bounds.

Signal x increases approximately with slope 2.5 during the next 100 time units:

$$On_{[0,100]}Max\ |x - 2.5T| - On_{[0,100]}Min\ |x - 2.5T| \leq 0.1$$

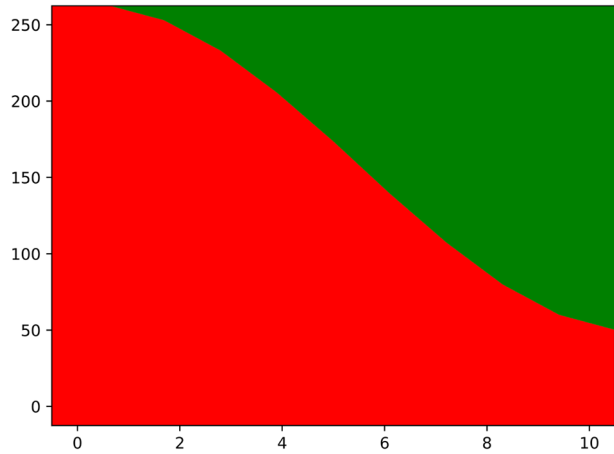
The previous expression can be rewritten in terms of derivatives in a much more simpler way:

$$On_{[0,100]}|Dx| \leq 2.5$$

Example 5 (Integral) Let us consider the following formula:

$$\varphi_f := On_{[0,p_1]}I\ x_0 \geq c_2 - p_2 \tag{1}$$

Fig. 4 Validity domain for signal $w_{10}(t)$ and the formula with integral



Consider the piece-wise constant signal w_{10} (Fig. 3) with sample values 0, 9 and 18 at times 0, 1 and 2 respectively and it ends at time 10. Let us now analytically compute the solution for the parametric identification problem for the above formula and the signal w_{10} . The value of the parameter p_1 varies between 0 and 10 and the parameter p_2 lies between 0 and 250 and c_2 equals 250. We can analytically compute the validity domain by considering 10 cases corresponding to the 10 segments of w_{10} . We give below the first three parts of the disjunction of constraint systems for representing the validity domain corresponding to the three cases which are $p_1 \in [0, 1)$, $p_1 \in [1, 2)$ and, $p_1 \in [2, 3)$:

$$\begin{aligned}
 region_1 &:= \{(p_1, p_2) \in \mathbb{R}^2 \mid (0 \leq p_1 < 1) \wedge (0 * p_1 \geq 250 - p_2)\} \\
 region_2 &:= \{(p_1, p_2) \in \mathbb{R}^2 \mid (1 \leq p_1 < 2) \wedge (0 * 1 + 9 * (p_1 - 1) \geq 250 - p_2)\} \\
 region_3 &:= \{(p_1, p_2) \in \mathbb{R}^2 \mid (2 \leq p_1 < 3) \wedge (0 * 1 + 9 * 1 + 18 * (p_1 - 1 - 1) \geq 250 - p_2)\} \\
 &\dots \\
 v_f &:= \bigcup_{i=1}^{10} region_i
 \end{aligned} \tag{2}$$

The full validity domain, named v_f , is computed as the union of all the regions and shown in green colour in Fig. 4. The validity domain for $(\neg(\neg\varphi_f))$ will be $(\neg(\neg v_f))$. If we naively compute this series of negations over semi-linear domains then the time complexity for obtaining the exact solution might be high. We implemented in C++ the negation operation on semi-linear domains using Parma Polyhedra Library [10]. We tested in addition to w_{10} using piece-wise constant signals w_{100} and w_{1000} each having 100 and 1000 segments respectively. We computed the validity domains by treating them as semi-linear domains for w_{10} , w_{100} and w_{1000} for $(\neg(\neg\varphi_f))$ while suitably setting the value of constant c_2 the ranges of values of parameters p_1 and p_2 . We observed computation times between 30 and 50 milliseconds for signal w_{10} and 57264 milliseconds for signal w_{100} in a conventional laptop. For w_{1000} the operations timed out after the one hour limit we set. We see that the computation time increases steeply with the size of the signal and becomes prohibitively high for signal w_{1000} which is of reasonable size.

On the other hand, if we reason in terms of membership queries to an oracle that guides the computation of the validity domains, then the previous $(\neg(\neg v_f))$ expression for signal w_{10} could be approximated efficiently (see Fig. 5) in 145 milliseconds. Our approach

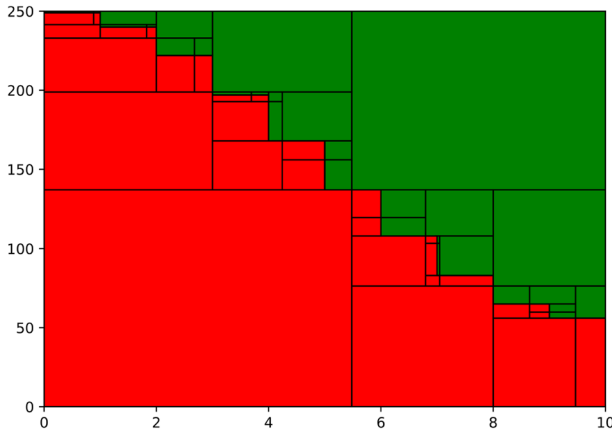


Fig. 5 Approximation of the validity domain for signal $w_{10}(t)$ and the formula with integral

requires 1997 and 75932 milliseconds respectively for approximating more than the 99% of the validity domains of signals w_{100} and w_{1000} . The C++ and ParetoLib code, as well as the w_{10} , w_{100} and w_{1000} signals in CSV format are publicly available.¹

More examples of properties that require quantitative STL operators can be found in the literature. For instance, the monitoring of the accumulated electricity consumption on smart meters is easily described with integrals over a time window [11]. The rest of the paper includes toy examples and a case study that amplifies the necessity of the new features we present. Further sections illustrate the methods we implement for approximating the validity domain we obtained in the last example, and outperform other monitoring tools that compute exact solution.

3 Extended signal temporal logic

3.1 Boolean semantics

Signal Temporal Logic (STL) [1] focuses on specifying properties of continuous-time signals that represent the evolution of attributes of a cyber-physical system. Formally, a signal is a function $\mathbf{x} : \mathbb{T} \rightarrow \mathbb{R}^n$, where the time domain $\mathbb{T} = [T_{\text{start}}, T_{\text{end}}]$ is a closed real interval, and the value $|\mathbf{x}| = T_{\text{end}} - T_{\text{start}}$ is the *duration* of the signal. We refer to signal components using their own letters: $x, y, z \dots \in \mathbb{T} \rightarrow \mathbb{R}$.

Then, the grammar of STL can be defined as follows:

$$\varphi = P(x_1, \dots, x_n) \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \mathcal{U}_{[t_1, t_2]} \varphi_2$$

- $P \in \mathbb{T} \rightarrow \{\text{true}, \text{false}\}$ is a predicate over the components of the input signal;

¹ <https://zenodo.org/doi/10.5281/zenodo.10156861>

- \mathcal{U} is the temporal operator *until*; $\varphi_1 \mathcal{U}_{[t_1, t_2]} \varphi_2$ asserts that φ_2 becomes true in the future, with the delay between t_1 and t_2 , and φ_1 must hold until that point.

The semantics of an STL formula φ with respect to a signal \mathbf{x} is its *satisfaction signal* $\llbracket \varphi \rrbracket \in \mathbb{T} \rightarrow \{\text{true}, \text{false}\}$; for every time point t , $\llbracket \varphi \rrbracket(t)$ tells whether or not φ holds at that time point. That is, $(\mathbf{x}, t) \models \varphi \iff \llbracket \varphi \rrbracket(t)$.

$$\begin{aligned} \llbracket P(x_1, \dots, x_n) \rrbracket(t) &= P(x_1(t), \dots, x_n(t)) \\ \llbracket \neg \varphi \rrbracket(t) &= \neg \llbracket \varphi \rrbracket(t) \\ \llbracket \varphi_1 \vee \varphi_2 \rrbracket(t) &= \llbracket \varphi_1 \rrbracket(t) \vee \llbracket \varphi_2 \rrbracket(t) \\ \llbracket \varphi_1 \mathcal{U}_{[t_1, t_2]} \varphi_2 \rrbracket(t) &= \exists t' \in [t + t_1, t + t_2]. \llbracket \varphi_2 \rrbracket(t') \wedge \forall t'' \in [t, t'] . \llbracket \varphi_1 \rrbracket(t'') \end{aligned}$$

Here, we use the non-strict semantics of *until* that requires the existence of a time point where both φ_1 and φ_2 hold.

Somewhat more commonly used temporal operators F (*eventually*) and G (*always*) can be defined in terms of *until*:

$$F_{[t_1, t_2]} \varphi \equiv \top \mathcal{U}_{[t_1, t_2]} \varphi \quad G_{[t_1, t_2]} \varphi \equiv \neg F_{[t_1, t_2]} \neg \varphi$$

$F_{[t_1, t_2]} \varphi$ asserts that φ becomes true with a delay between t_1 and t_2 ; and $G_{[t_1, t_2]} \varphi$ asserts that for every time point between t_1 and t_2 time points in the future, φ holds.

$$\begin{aligned} \llbracket F_{[t_1, t_2]} \varphi \rrbracket(t) &= \exists t' \in [t + t_1, t + t_2]. \llbracket \varphi \rrbracket(t') \\ \llbracket G_{[t_1, t_2]} \varphi \rrbracket(t) &= \forall t' \in [t + t_1, t + t_2]. \llbracket \varphi \rrbracket(t') \end{aligned}$$

Note that the atomic propositions of STL are predicate signals; they are assumed to be produced from sensor readings of some cyber-physical system, but the logic does not specify how exactly this happens. The satisfaction signals are piecewise-constant (since the range is $\{\text{true}, \text{false}\}$), and this is the only kind of signals STL monitors need to work with.

3.2 Quantitative semantics: new additions to STL

STL also admits robustness semantic that measures how far is an observed behaviour from satisfying/violating a specification. In this case, the semantics of a formula φ is interpreted as a piecewise-constant or piecewise-linear function from real-time, i.e., φ formula is a function $\llbracket \varphi \rrbracket : \mathbb{T} \rightarrow \mathbb{R}$. Thus, φ has real-valued switching points. This feature facilitates the inclusion of quantitative operators while treating real numbers 0 and 1 as Booleans (i.e., $0 \equiv \text{false}$, $1 \equiv \text{true}$). The quantitative interpretation of a φ formula involves that Boolean expressions are rewritten in terms of the composition of min/max operators:

$$\varphi_1 \wedge \varphi_2 \equiv \min\{\varphi_1, \varphi_2\} \quad \varphi_1 \vee \varphi_2 \equiv \max\{\varphi_1, \varphi_2\} \quad \neg \varphi \equiv 1 - \varphi$$

This quantitative interpretation should not be confused with the quantitative semantics for STL called space robustness (known popularly as simply robustness) defined in [12]. This robustness denoted as ρ indicates by how much margin a given signal w satisfies an STL formula φ . It follows that positive and negative robustness values correspond to satisfaction and violation of the formula respectively. One can now see why while our quantitative interpretation requires $\neg \varphi \equiv 1 - \varphi$, robustness changes sign when negation is applied over a formula i.e. $\rho(\neg \varphi, w) = -\rho(\varphi, w)$. Similarly to φ for STL in Boolean semantics, a φ formula may refer to an input signal \mathbf{x} ; apply a real-valued function f pointwise to the outputs

its φ subformulas; be a temporal operator (\mathcal{U}, F, G); or, additionally, apply an aggregate function over the sliding window ($On_{[a,b]}$). We define it as:

$$\begin{aligned} \llbracket x \rrbracket(t) &= x(t) & \llbracket On_{[a,b]}\psi \rrbracket(t) &= \llbracket \psi \rrbracket([t + a, t + b]) \\ \llbracket f(\varphi_1 \dots \varphi_n) \rrbracket(t) &= f(\llbracket \varphi_1 \rrbracket(t) \dots \llbracket \varphi_n \rrbracket(t)) \end{aligned}$$

We abuse the notation so that x is both a symbol referring to a component of an input signal and the corresponding real-valued function; similarly, f is both a function symbol and the corresponding function.

The necessity of expressing richer properties in STL motivates the incorporation of STL dialects to ParetoLib that extend the original set of operators with additional quantitative primitives. As previously sketched in the motivating examples, these predicates compute, for instance, the minimum/maximum values of a signal in a time window [3], the value of the derivative in a time point or the integral in a time interval [13], or counts the number of events in a sequence [5]. Section 5 will show a case study that requires the count operator for learning patterns in labelled electrocardiograms (ECG). The following grammar summarises the list of operators that are supported by ParetoLib 2.0:

$$\begin{aligned} \varphi &:= c_e \mid x \mid f(\varphi_1 \dots \varphi_n) \mid f_+^d(\varphi) \mid f_-^d(\varphi) \mid On_{[t_1,t_2]}\psi \mid \varphi_1 \mathcal{U}_{[t_1,t_2]}^d \varphi_2 \\ \psi &:= Min \varphi \mid Max \varphi \mid \int^t \varphi \end{aligned}$$

Min, Max. Operations like minimum/maximum or integrals aggregate the values for a time window. The semantics of a ψ formula is a function $\llbracket \psi \rrbracket : (\mathbb{R} \cup -\infty) \times (\mathbb{R} \cup \infty) \rightarrow \mathbb{R}$ from an interval of time with real lower bound to a dual value. A ψ formula is evaluated on an interval and does not have an output signal by itself. Instead, it supplies an aggregate operation that will be computed when evaluating the containing *On* formula or *until* formula. It should be possible to efficiently compute this aggregate operation over a sliding window, and it should preserve the chosen shape of signals.

Since we focus on piecewise-constant and piecewise-linear signals, the two operations that we can immediately offer are *Min* and *Max*, which can be efficiently computed over a sliding window using the algorithm of D. Lemire [14, 15], and preserve the piecewise-constant and piecewise-linear shapes.

$$\llbracket Min \varphi \rrbracket[t_1, t_2] = \min_{[t_1,t_2]} \llbracket \varphi \rrbracket \quad \llbracket Max \varphi \rrbracket[t_1, t_2] = \max_{[t_1,t_2]} \llbracket \varphi \rrbracket$$

At this point, we extend *until* operator in order to align with the quantitative semantics. For every time point t , we either associate an interval ending when φ_2 becomes non-zero (i.e., starts holding); or we report that no suitable end point was found. When such interval exists, we evaluate φ_1 on it. When the interval does not exist, we produce d . Formally:

$$\llbracket \varphi_1 \mathcal{U}_{[t_1,t_2]}^d \varphi_2 \rrbracket(t) = \begin{cases} \llbracket \varphi_1 \rrbracket_{[t',t]}, & \text{if } \exists t' \in [t + t_1, t + t_2], \text{ s.t. } \llbracket \varphi_2 \rrbracket(t') \neq 0 \\ d, & \text{otherwise} \end{cases}$$

Standard STL *eventually* and *always* operators can be expressed in the new language as follows:

$$F_{[t_1,t_2]}\varphi \equiv On_{[t_1,t_2]}Max \varphi \quad G_{[t_1,t_2]}\varphi \equiv On_{[t_1,t_2]}Min \varphi$$

Finally, standard STL *until* operator (denoted as \mathcal{U}^{STL}) is expressed as follows:

$$\varphi_1 \mathcal{L}_{[t_1, t_2]}^{STL} \varphi_2(t) \equiv (\text{Min } \varphi_1) \mathcal{L}_{[t_1, t_2]}^0 \varphi_2$$

ϵ -count (c_ϵ). The ϵ -count operator counts the number of *positive* events in a signal (i.e., the moments when a Boolean signal becomes true). The ϵ -count operator is not explicitly available at the STL grammar level, but it can be used for counting the number of positive intervals in a Boolean signal via the *OracleEpsSTLe* wrapper in ParetoLib. The ϵ -count operator grounds on the *support* of a Boolean signal \mathbf{x} , denoted by $\text{supp}(\mathbf{x})$, which is the topological closure of $\{t \mid \mathbf{x}(t) \neq 0\}$, that is, the smallest closed set that contains all the points where the signal is not false. The ϵ -count operator, originally defined in [5], is inspired by the notions of ϵ -separated sets and ϵ -capacity proposed in [16]. The ϵ -count operator contrasts to the Lebesgue’s measure of subsets of \mathbb{R} which entails a small measure for a signal whose support is the disjoint union of many intervals of almost-null measure which are quite far apart. Formally speaking, the ϵ -count operator is defined as:

Definition 1 (ϵ -separated set and ϵ -count) Given a Boolean signal \mathbf{x} , a set S of reals is ϵ -separated w.r.t. \mathbf{x} if $S \subseteq \text{supp}(\mathbf{x})$ and for every $t, t' \in S$ with $t \neq t'$, it holds that $|t - t'| \geq \epsilon$. The ϵ -count of a signal \mathbf{x} is $c_\epsilon(\mathbf{x}) = \max\{|S| \mid S \text{ is } \epsilon\text{-separated w.r.t. } \mathbf{x}\}$.

The ϵ -count of a signal \mathbf{x} is determined in a greedy manner with the following recursive equations:

- $c_\epsilon(\mathbf{0}) = 0$ when it is applied to the constant signal $\mathbf{0}$, and
- $c_\epsilon(\mathbf{x}) = 1 + c_\epsilon(\mathbf{x}')$ where $\mathbf{x}'(t) = 0$ if $t < \epsilon + \min(\text{supp}(\mathbf{x}))$ and $\mathbf{x}'(t) = \mathbf{x}(t)$ otherwise.

Finally, the ϵ -count has some remaining properties:

- (1) The ϵ -count is null iff it is applied to the constant signal $\mathbf{0}$.
- (2) The ϵ -count is increasing: if $\mathbf{x} \leq \mathbf{x}'$ then $c_\epsilon(\mathbf{x}) \leq c_\epsilon(\mathbf{x}')$.
- (3) The ϵ -count satisfies a triangular inequality: $c_\epsilon(\mathbf{x} \vee \mathbf{x}') \leq c_\epsilon(\mathbf{x}) + c_\epsilon(\mathbf{x}')$.

Derivatives, Integrals. The derivatives and integrals are calculated using the following mathematical expressions, where \mathbf{x} stands for the complete signal, \mathbf{x}_τ is the value of the signal at time τ , $f_{[t_1, t_2]}^i(\varphi)^2$ is the integral for the time interval $[t_1, t_2]$ and $f_+^d(\varphi)$ ($f_-^d(\varphi)$) represent the right (left) derivatives:

$$\begin{aligned} \llbracket f_+^d(\varphi) \rrbracket &= \frac{df(\varphi)}{dt^+} & \llbracket f^i(\varphi) \rrbracket &= \int f(\varphi_\tau) \delta\tau \\ \llbracket f_-^d(\varphi) \rrbracket &= \frac{df(\varphi)}{dt^-} & \llbracket f_{[t_1, t_2]}^i(\varphi) \rrbracket &= \int_{t_1}^{t_2} f(\varphi_\tau) \delta\tau \end{aligned}$$

Due to the implicit signal discretization, \mathbf{x}_τ , the value of the input signal at time τ , is obtained as $\mathbf{x}_{k\delta t}$ where k is the k -th term of the array, δt is the time step determined by the sampling frequency and $k\delta t \leq \tau < (k + 1)\delta t$. The quantitative measurements of derivatives and integrals are then discretized and approximated by the following equations:

² $f_{[t_1, t_2]}^i(\varphi)$ is a syntax sugar for $\text{On}_{[t_1, t_2]} f^i(\varphi)$

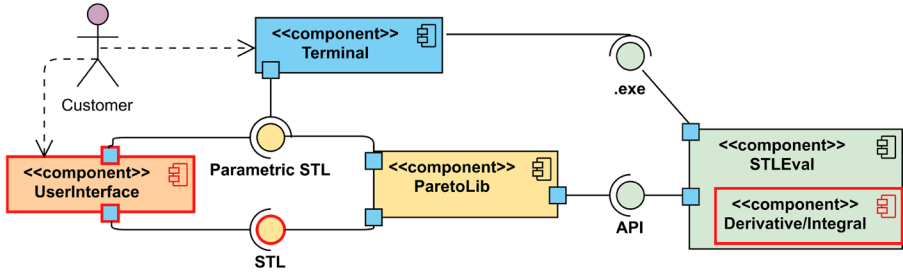


Fig. 6 Internal structure of ParetoLib 2.0

$$\begin{aligned} \llbracket f_+^d(\mathbf{x}_\tau) \rrbracket &= \frac{f(\mathbf{x}_{(k+1)\delta t}) - f(\mathbf{x}_{k\delta t})}{\delta t} & \llbracket f_{[a,b]}^i(\mathbf{x}) \rrbracket &= \sum_{k'=k+a/\delta t}^{k+b/\delta t-1} f(\mathbf{x}_{k'\delta t})\delta t \\ \llbracket f_-^d(\mathbf{x}_\tau) \rrbracket &= \frac{f(\mathbf{x}_{k\delta t}) - f(\mathbf{x}_{(k-1)\delta t})}{\delta t} \end{aligned}$$

4 Tool presentation

4.1 Internal structure

The internal structure of ParetoLib 2.0 is depicted in Fig. 6. The components that are highlighted in red represent the parts that are updated or completely new with respect to the previous releases of ParetoLib.

The architecture of ParetoLib 2.0 is divided into three parts: the GUI, the STL monitor and the modules for mining the parameter valuations in parametric specifications. Depending on the type of STL specification the user introduces in the GUI (parametric vs non-parametric STL properties), ParetoLib directly forwards the query to the STL monitor or starts the mining procedure for extracting the validity domain. In any case, ParetoLib uses external STL monitors as oracles for answering STL queries. ParetoLib 2.0 continues to rely on top of StlEval [17], a C++ engine for evaluating extended Signal Temporal Logic (STL) specifications. ParetoLib interfaces to StlEval with a C API that includes the new quantitative operators (e.g, derivatives or integrals over the signals). Other STL monitors such as AMT 2.0 [18] are also supported via Python code (replacing OracleSTLELib by OracleSTL in Listings 1) or can be easily incorporated thanks to the modularity of the library. As the features of AMT 2.0 have not been enlarged since the previous release of ParetoLib, this STL monitor is omitted from the previous UML diagram for the sake of concision.

The functionalities for the internal modules of ParetoLib remain unchanged. These modules are responsible for learning the parameter valuations of parametric STL specifications. The mining algorithm converts the PSTL formulas into numerical STL instances that queries to StlEval [19]. The current version of ParetoLib optimises the performance of this process by adding support to multi core CPU and type annotations and Cython directives [20] to compile the Python modules into C code during the installation phase. The rest of the code is also updated in order to support recent Python versions and standards.

The graphical user interface (GUI) allows to interact both with the mining library for parametric STL specifications and indirectly with the StlEval tool for numerical STL properties. The GUI simplifies the interaction with StlEval, but experienced users can also run StlEval through the OS terminal for evaluating non-parametric STL specifications. Similarly, ParetoLib 2.0 can also be imported as a Python library and run in the Python console, as illustrated by the code in Listings 1.

The versatility of ParetoLib 2.0 adapts the complexity of the tool to the user level: inexperienced users can interact with STL monitors via the GUI, while programmers and software engineers can code scripts and use ParetoLib 2.0 as a Python library.

```

1 from ParetoLib.Oracle.OracleSTLe import OracleSTLeLib
2 from ParetoLib.Search.Search import Search2D, EPS, DELTA, STEPS
3
4 # File containing the definition of the Oracle (i.e., the Parametric STL
5 # spec)
6 nfile = '../.../Tests/Oracle/OracleSTLe/2D/triangular/derivative/
7     triangular.txt'
8 human_readable = True
9
10 # Definition of the parameters space.
11 # In this example, there are two parameters ranging in [1.0, 999.0] for p1
12 # and [0.0, 2.0] for p2.
13 min_x, min_y = (1.0, 0.0)
14 max_x, max_y = (999.0, 2.0)
15
16 # Initialization of the Oracle from the configuration file 'nfile'.
17 oracle = OracleSTLeLib()
18 oracle.from_file(nfile, human_readable)
19
20 # Mining the parameter valuation of p1 and p2 that verifies/falsifies the
21 # parameter space.
22 # The variables (EPS, DELTA, STEPS) tune the accuracy of the computations.
23 rs = Search2D(ora=oracle,
24             min_cornerx=min_x, min_cornery=min_y,
25             max_cornerx=max_x, max_cornery=max_y,
26             epsilon=EPS, delta=DELTA, max_step=STEPS,
27             sleep=0, opt_level=0,
28             blocking=False, parallel=False,
29             logging=True, simplify=True)
30 rs.to_file("result.zip")

```

Listing 1: Running the ParetoLib library in a Python script.

ParetoLib 2.0 provides an all-in-one bundle installer that ships the GUI as well as the DLL and binaries of StlEval for Windows and Linux. Except for the STL monitors (StlEval and AMT), ParetoLib is mainly written in Python. Hence, ParetoLib 2.0 becomes a cross-platform tool and can interact with the rest of the Python ecosystem (e.g., it can be easily imported as a library in a Jupyter Notebook).

4.2 StlEval

Given a non-parametric STL specification, StlEval implements efficient monitors that evaluate the properties on a real-time signal. StlEval has been upgraded in order to natively process new quantitative operators that count the number of events (ϵ -count) on a signal or calculate the derivatives and integrals. Previously, StlEval already included operators for computing other quantitative properties such as the maximum/minimum values of a signal in a time window.

StlEval receives a finite representation of the continuous real-time signal in the format of a *comma separated values* (CSV) array, which is interpreted as a constant piecewise function. Each row of the CSV file has a pair $\langle \text{timestamp}, \text{value} \rangle$. Timestamps are integer values starting in 0, and the samples are floating numbers. Next, StlEval reads a text file with the STL specification. StlEval uses prefix notation, i.e., $(F_{[0,p_1]}(\leq (\text{abs}(D \mathbf{x}_0)) p_2))$.

4.3 Mining algorithms in ParetoLib

ParetoLib converts the problem of mining parametric STL equations into solving a multi criteria optimisation problem. Our tool provides three algorithms for computing the Pareto front that will return the valuations that satisfy the parametric STL expressions. All the methods replace the variables in PSTL equations by numerical STL instances that are query to an external STL monitor. Based on the Boolean answer the STL monitor replies, the mining algorithms guide the discovery of the validity domain.

The first method, named BBMJ19 [4], calculates a single Pareto front while the second method, named BDMJ20 [5], obtains the intersection of two Pareto fronts. The BBMJ19 method was originally defined in [21, 22] and shipped in the first release of ParetoLib in [4], while method BDMJ20 was incorporated in later versions. These algorithms assume that the STL properties are monotonic and use the concepts of *upset*, *downset* and *Pareto front*. Recently, method BMNN23 [7] has also been incorporated to ParetoLib, which allows for mining parametric STL specifications that do not require monotonicity assumptions of the validity domain. It provides two types of algorithms that partition the search space of a PSTL property into smaller boxes and compute the validity domain for the parameter valuations based on statistical constraints.

The computational costs of BBMJ19, BDMJ20 and BMNN23 depend on the accuracy requirements for the computation of the validity domain. For instance, BBMJ19 categorises the search space into (in)valid and unexplored area. The volume contained in the unexplored area decreases below $\delta\%$ of the initial volume V_0 in $\mathcal{O}(V_0/\delta^{\kappa_m 2^{m-1}})$ where m is the number of parameters (or dimensions) in a PSTL formula and κ_m is a constant number in the interval $[2m - 4, 2^m - 3]$ which represents the number of boxes created during the partitioning step.

BMNN23 relies on a partition strategy which can use a fixed or dynamic cell size. In the worst case, it costs $\mathcal{O}(M^m) \cdot \mathcal{O}(n)$, where m is the number of parameters (or dimensions) in a PSTL formula, and M is a constant that depends on the granularity of the cell size. As BMNN23 uses statistical properties for partitioning the validity domain, the method requires n input signals or samples.

Finally, the evaluation of non-parametric STL properties is commonly handled by STL monitors in linear time with respect to the length of the input signal.

Partial order on \mathbb{R}^n , Upset, Downset and Pareto Front. Given two vectors $p, q \in \mathbb{R}^n$, vector p is lower than q , denoted by $p \leq q$, if $\forall i, p_i \leq q_i$. A set \bar{X} is an *upset* if for all $p, q \in \mathbb{R}^n$ such that $p \leq q$ if $p \in \bar{X}$ then $q \in \bar{X}$. A set \underline{X} is a *downset* if for all $p, q \in \mathbb{R}^n$ such that $q \leq p$ if $p \in \underline{X}$ then $q \in \underline{X}$. The boundary consisting of all the minimal elements of an upset (or all the maximal elements of a downset) is called a *Pareto front* in the field of multi-criteria optimisation. The *box* between two vectors \underline{x} and \bar{x} with $\underline{x} \leq \bar{x}$ is $[\underline{x}, \bar{x}] = \{y \mid \underline{x} \leq y \leq \bar{x}\}$.

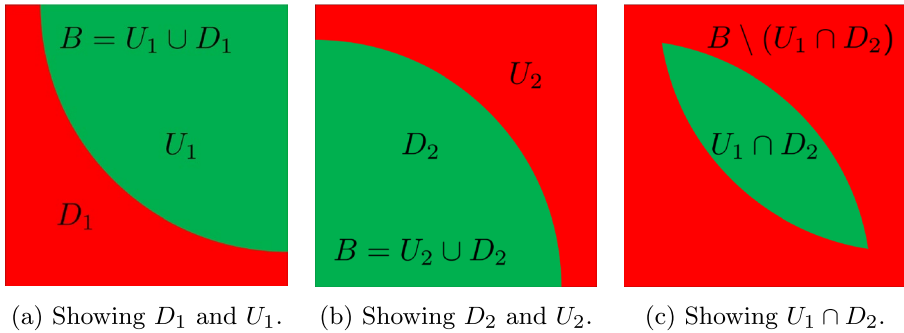


Fig. 7 Illustration for Pareto algorithms

Both algorithms in ParetoLib deal with user-given boxes. Consider the box B from Fig. 7. The first Pareto front P_1 separates the upset U_1 and the downset D_1 (Fig. 7a), while the second Pareto front P_2 separates the upset U_2 and the downset D_2 (Fig. 7b). The algorithms rely on the existence of two oracle functions defined over the box B . Given a query point x contained in B , the first oracle function $o_1(x)$ indicates whether x belongs to U_1 or not (i.e., $x \in U_1$). The second oracle function $o_2(x)$ works similarly and returns the membership of x to D_2 (i.e., $x \in D_2$).

Assuming the existence of a Pareto front, the BBMJ19 algorithm takes an oracle and a box to approximately compute the upset and downset. In particular, U_1 and D_1 are approximated using oracle o_1 and box B . Similarly, U_2 and D_2 are computed using oracle o_2 and box B . Please note that BBMJ19 requires that the oracles are monotonic. In these examples, o_1 is increasing: for $p, q \in B$ with $p \leq q$, it applies that $o_1(q) \geq o_1(p)$. Similarly, the oracle o_2 is decreasing: for $p, q \in B$ with $p \leq q$, it applies that $o_2(q) \leq o_2(p)$.

Conversely, the algorithm BDMJ20 computes the intersection of two Pareto fronts. This method applies when we are dealing with one increasing oracle and another decreasing oracle: it involves the intersection of the upset of the increasing oracle and the downset of the decreasing oracle. For example, it can approximately compute $U_1 \cap D_2$ (in Fig. 7c) using oracles o_1, o_2 and the box B . It computes approximately but directly, the intersection of U_1 and D_2 . A less efficient way is to separately compute U_1 and D_2 and then intersect them.

4.4 Graphical user interface

The main GUI window (Fig. 8) is implemented using PyQt5. Extra libraries such as Matplotlib, Pandas and Seaborn help for displaying the time series or the parameter valuation results (Figs. 11, 12 and 13). The GUI starts when running the command `python ParetoLib/GUI/GUI.py`.

The tool receives three input files: (1) A finite representation of the continuous signal, (2) A (parametric) STL specification, and (3) Optionally, the list of parameter names. Signal files are in CSV format, and the rest of input files are in textual format with `.stl/param` extension. These files must comply with the `StlEval` format. In future versions, we aim at adding a selector which chooses the monitoring tool directly in the user interface (i.e., `StlEval` or `AMT`).

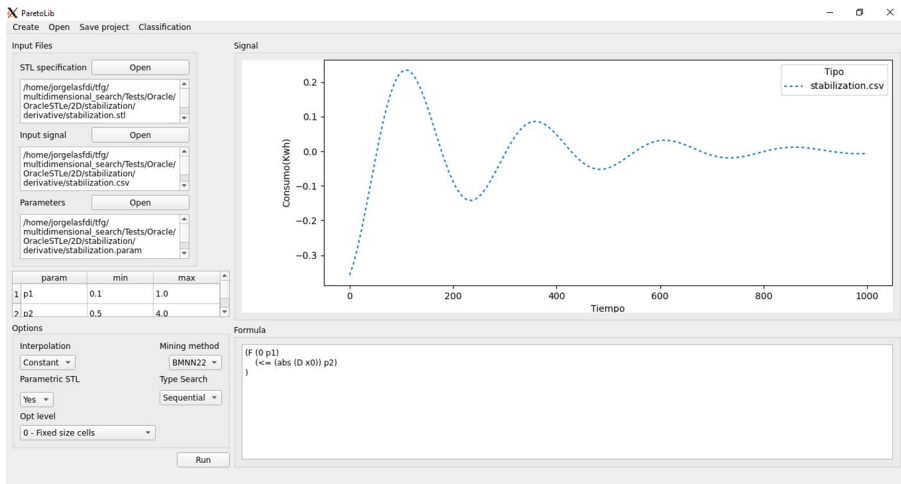


Fig. 8 Main window of ParetoLib 2.0

Then, the user chooses the values of the parameter range (if any) and configure the execution in the *option* area. For instance, the user can select the type of interpolation, whether the STL specification is parametric or not, or the mining algorithm in case it is parametric (mining algorithms are BBM19 [4], BDMJ20 [5] or BMNN23 [7]). In case of running the BDMJ20 method, the user must provide two STL specification files: one for each oracle, as described in Sect. 4.3.

Running the evaluation of a non-parametric STL formula opens a pop-up window with a message saying if the property is satisfied or violated. If ParetoLib receives a parametric STL formula, it learns the parameter valuations and returns a window that plots the results (Figs. 11, 12 and 13).

Finally, the GUI offers a menu bar with useful options. For instance, it includes buttons for creating, loading and saving ParetoLib projects in a JSON file: this file will contain references to the input file paths and current configuration (e.g., mining method). In case that we desire to compare several Pareto fronts, the *classification* button will show the most characteristic element of the green region for each validity domain. This feature requires a list of validity domains in ParetoLib file format (see line 26 in Listing 1). Then, it internally computes the Hausdorff distance among the set of points.

4.5 Example

Using the new version of StIEval, ParetoLib can solve non-parametric and parametric formulas involving quantitative operators via the GUI or the Python terminal. As previously stated in the motivating examples in Sect. 2, some properties in cyber-physical systems are easier expressed in terms of these quantitative primitives than using other STL approaches. Let's consider again two common scenarios: decaying signals (Fig. 9) and periodic signals (Fig. 10). Detecting whether the signal stabilises or not is equivalent to check that the wave is almost flat, i.e., the derivative bounds are near zero in the

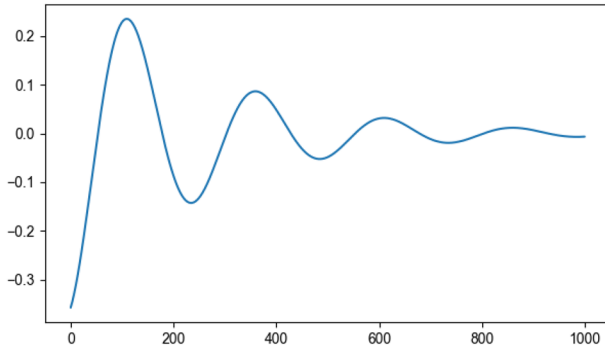


Fig. 9 Decaying signal

Fig. 10 Triangular signal with 4000 time units period

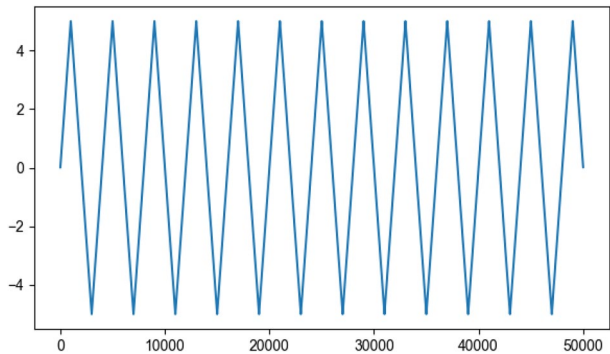
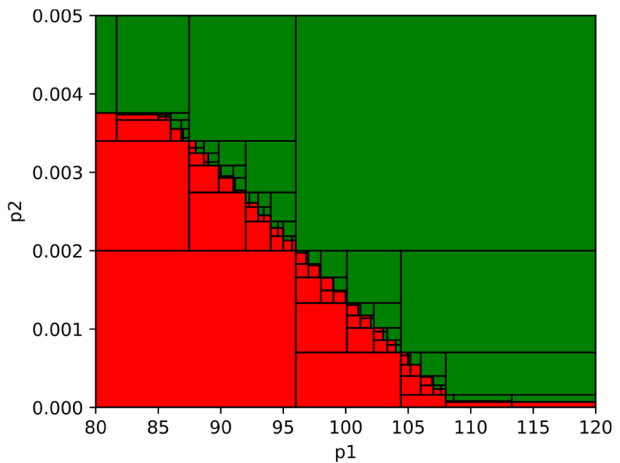


Fig. 11 Parameter valuation result for Eq. (3).



future. The usage of a parametric STL expression allows the inspection of the stabilisation property over a period of time and oscillation amplitudes:

Fig. 12 Boolean signal resulting from substitution of $(p_1, p_2) = (100, 0.05)$ in Eq. (3)

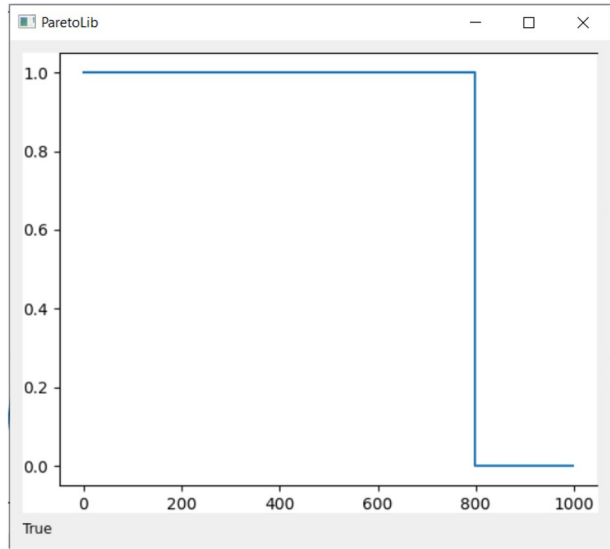
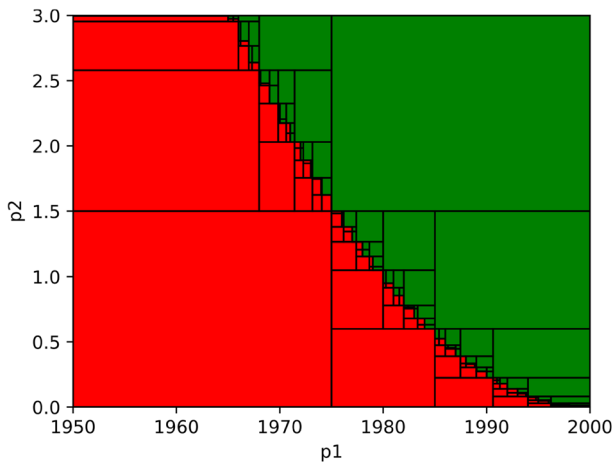


Fig. 13 Parameter valuation result for Eq. (4)



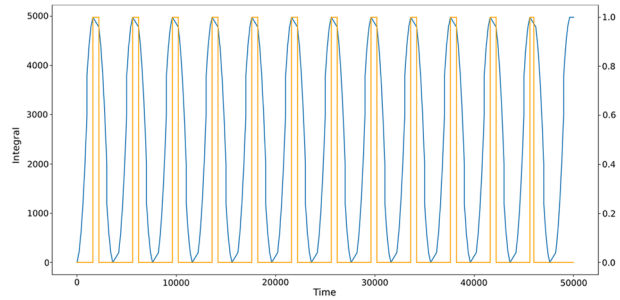
$$F_{[0,p_1]} |D \mathbf{x}_0| \leq p_2 \tag{3}$$

The parameters in the PSTL equation are p_1 and p_2 , \mathbf{x}_0 is the first component of the input signal, D is the derivative operator and $|\cdot|$ computes the absolute value.³ The parameter p_1 corresponds to the upper time bound of the sliding window, and p_2 corresponds to the oscillation amplitude.

As a result, ParetoLib returns the parameter valuation (Fig. 11) that shows the combinations of parameters p_1 and p_2 that satisfy the property (green region) or falsify it (red region). Figure 12 shows the Boolean signal produced replacing p_1 and p_2 by 100 and 0.05 in Eq. (3). It also textually reports if the STL property is satisfied at time instant 0 (i.e., True).

³ For the sake of readability, the equation uses infix notation.

Fig. 14 Boolean signal resulting from substitution of $(p_1, p_2) = (4000, 50)$ in Eq. (4)



Similarly, an example of parametric expression involving integrals is Eq. (4). In this case, $On_{[0,p_1]}$ refers to the integration interval for the I operator. If p_1 is multiple of the periodicity of oscillation of the signal around $\mathbf{x}_0(t) = 0$, then the integral should also be zero. Figure 13 shows the parameter valuation for Eq. (4) when evaluated over the triangular signal (Fig. 10). Figure 14 shows the integral of the triangular signal (blue) over time. The Boolean signal (orange) is produced replacing p_1 and p_2 by 4000 and 50 in Eq. (4).

$$On_{[0,p_1]} I \mathbf{x}_0 \geq 5000 - p_2 \quad (4)$$

The files for running the previous examples are located in the *Test* folder of the git repository of ParetoLib. These files include the STL specifications, the signals and a file containing the names of the parameter variables. Depending on the monitoring tool (i.e., StlEval or AMT), the file format of the STL specification and signal may change. Additional examples are located in the *doc* folder, as well as a video demo.

5 Case study

The goal of this section is to illustrate the new features of ParetoLib. In particular, we will show the benefits of the quantitative operators for simplifying the specification of STL properties as well as the computational speed up compared to the first release of ParetoLib.

In this case study, we design a *disease detector* for electrocardiograms (ECG) using parametric STL. Informally speaking, an ECG is composed of a sequence of approximately flat regions followed by a pulse or heart beat. If the patient suffers from arrhythmia or other kind of heart diseases, the pulse exhibits an anomalous shape. In the following experiments, we use the MIT-BIH Arrhythmia Database of Physionet [23, 24], a public library that contains several annotated ECG's with thousands of pulses per sample. A portion of ECG 221 is depicted as in Fig. 15 where the signal (blue) and the annotations (orange) come from the database. The annotations for the normal peaks are modelled into a labelling signal that is 1 when a normal peak occurs and 0 everywhere else.

Our aim is to develop a pattern predictor (i.e., classifier) that identifies normal peaks and mimic the labelling a doctor would manually make for an input ECG. The learning procedure presented in [5] relies on the ϵ -count operator for mining a classifier that minimises the number of mismatches (either false positives, f_+ , or false negatives, f_-) with respect to the annotations of the samples in the training set. The inferred classifier consists of a PSTL formula and the solution of the parameter space. Obtaining a classifier that mimics a doctor labelling is

hence translated into an optimisation problem that minimises both f_+ and f_- . The problem is solvable by the BDMJ20 method in Sect. 4.3 that deals with two, maybe opposing, optimisation criteria.

Firstly, let’s introduce some formal definitions and notation. A labelled signal (s, λ_s) is a pair of signal s and labelling signal λ_s . We use \mathcal{S} to denote the given set of labelled signals. We use IPPP to mean Increasing Parametric Pattern Predictor (defined formally in [5]). To put it in simple terms, increasing the parameter values for an IPPP makes the true predictions become more common. We aim at learning parameters p for an IPPP Ψ_p so that for every given labelled signal (s, λ_s) , the labelling signals $\Psi_p(s)$ and λ_s should match together as much as possible. We measure two kind of mismatches by measuring “how often” the two following signals are true. The *false positive signal* $\neg\lambda_s \wedge \Psi_p(s)$ indicates when the predictor predicts an occurrence when there is none. The *false negative signal* $\lambda_s \wedge \neg\Psi_p(s)$ indicates when the predictor misses an actual occurrence.

Given bounds f_+, f_- on the allowed ϵ -count of false positives and false negatives, we are interested in the following three sets:

$$\text{Dom}+(\Psi, \mathcal{S}, f_+) = \{p \mid \forall (s, \lambda_s) \in \mathcal{S}, c_\epsilon(\Psi_p(s) \wedge \neg\lambda_s) \leq f_+\}, \tag{5}$$

$$\text{Dom}-(\Psi, \mathcal{S}, f_-) = \{p \mid \forall (s, \lambda_s) \in \mathcal{S}, c_\epsilon(\neg\Psi_p(s) \wedge \lambda_s) \leq f_-\}, \tag{6}$$

$$\text{DomInter}(\Psi, \mathcal{S}, f_+, f_-) = \text{Dom}+(\Psi, \mathcal{S}, f_+) \cap \text{Dom}-(\Psi, \mathcal{S}, f_-). \tag{7}$$

For convenience, we call them respectively the *positive*, *negative* and *intersection* solution sets. In addition, we are interested in a relaxed version of the identification problem for false positive bounding, by tolerating a difference of σ time units in matching the labels. This can be done by replacing λ_s with the signal⁴ $F_{[-\sigma, \sigma]} \lambda_s$ in (5). More concretely, the solution set of the corresponding σ -relaxed problem is:

$$\text{Dom}^{+\sigma}(\Psi, \mathcal{S}, f_+) = \{p \mid \forall (s, \lambda_s) \in \mathcal{S}, c_\epsilon(\Psi_p(s) \wedge \neg F_{[-\sigma, \sigma]} \lambda_s) \leq f_+\}.$$

Hence, the corresponding relaxed version of the intersection solution set (7) is

$$\text{DomInter}^\sigma(\Psi, \mathcal{S}, f_+, f_-) = \text{Dom}^{+\sigma}(\Psi, \mathcal{S}, f_+) \cap \text{Dom}-(\Psi, \mathcal{S}, f_-). \tag{8}$$

Note that $\text{Dom}+(\Psi, \mathcal{S}, f_+)$ and $\text{Dom}^{+\sigma}(\Psi, \mathcal{S}, f_+)$ are downsets and $\text{Dom}-(\Psi, \mathcal{S}, f_-)$ is an upset because Ψ is increasing (recall definitions of upset and downset from Sect. 4.3). The sets $\text{DomInter}(\Psi, \mathcal{S}, f_+, f_-)$ and $\text{DomInter}^\sigma(\Psi, \mathcal{S}, f_+, f_-)$ are intersections of an upset and a downset, which we can compute from the intersection of two Pareto fronts using BDMJ20 algorithm.

It is also of great interest to compute the set of couples (f_+, f_-) , called *set of feasible error bounds*, for which a solution exists:

$$\mathcal{P}(\Psi, \mathcal{S}) = \{(f_+, f_-) \mid \text{DomInter}^\sigma(\Psi, \mathcal{S}, f_+, f_-) \neq \emptyset\}. \tag{9}$$

The set $\mathcal{P}(\Psi, \mathcal{S})$ is an upset and its minimal elements form a Pareto front. We compute it via membership-queries for couples (f_+, f_-) . They are done via non-emptiness checking of $\text{DomInter}^\sigma(\Psi, \mathcal{S}, f_+, f_-)$ which is an easier problem than computing the whole set.

⁴ where $(F_{[-\sigma, \sigma]} \lambda_s)(t) = 1$ iff $\exists t' \in [t - \delta, t + \delta], s(t') = 1$.

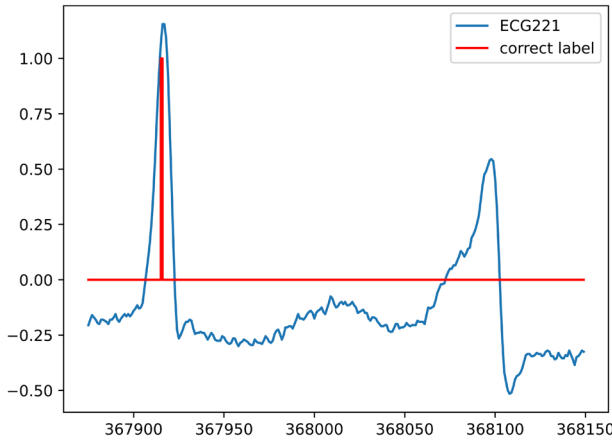


Fig. 15 Excerpt from ECG221

Equation (10) gives a simple and rough characterisation of a normal ECG peak using min/max operators of extended STL [3]. For the sake of readability, the syntax has been slightly simplified by replacing $On_{[a,b]}Max \mathbf{x}$ by $Max_{[a,b]} \mathbf{x}$. Equation (10) is equal to 1 if the maximum of \mathbf{x} on $[t - p_1, t + p_1]$ is above $-p_3$, and its variation is within the bound p_2 on $[t - c, t - p_1]$ and on $[t + p_1, t + c]$. The parameter domains are $p_1 \in [0, 70]$, $p_2 \in [0, 1]$ and $p_3 \in [-1, 0]$. Here, $c = 70$ is a constant representing an upper limit on p_1 . Note that if one increases p_1 , p_2 or p_3 , the property is easier to achieve. Alternatively, Eq. (11) characterises ECG pulses in terms of derivatives and integrals in a simpler and more readable way.

$$\Psi_{(p_1, p_2, p_3)}^{ch_1}(\mathbf{x})(t) := ((Max_{[-c, -p_1]} \mathbf{x} - Min_{[-c, -p_1]} \mathbf{x}) \leq p_2) \wedge ((Max_{[p_1, c]} \mathbf{x} - Min_{[p_1, c]} \mathbf{x}) \leq p_2) \wedge ((Max_{[-p_1, p_1]} \mathbf{x}) \geq -p_3) \tag{10}$$

$$\Psi_{(p_1, p_2, p_3)}^{ch_2}(\mathbf{x})(t) := (|Dx| < 1) \mathcal{U}_{[0, p_1]} (I_{[0, p_2]} x) < p_3 \tag{11}$$

For ECG 221, the predictor in Eq. (10) can match the labelling with no false negatives ($f_- = 0$) and only a single false positive ($f_+ = 1$): it identifies the blue peak in the right hand side of Fig. 15 as a valid pulse. We denote the singleton signal set containing the labelled ECG 221 signal as \mathcal{S}_{221} . Table 1 shows the execution time that ParetoLib requires for computing the solution set ($DomInter^\sigma(\Psi^{ch_1}, \mathcal{S}_{221}, f_+ = 1, f_- = 0)$) of the parametric space (p_1, p_2, p_3) with different accuracy. The value of V_δ represents the percentage of the parametric space that remains unexplored. For instance, the green area in Fig. 16 represents the solution set of the parameters when the unexplored area is less than 0.01%.

Table 1 Execution time (seconds) of the solution set of the parametric space for different scenarios

ECG 221	$V_\delta = 1\%$	$V_\delta = 0.1\%$	$V_\delta = 0.01\%$
Python	97.66	403.58	1974.56
Cython	93.66	391.22	1940.21
Multicore Python	55.05	185.67	861.70

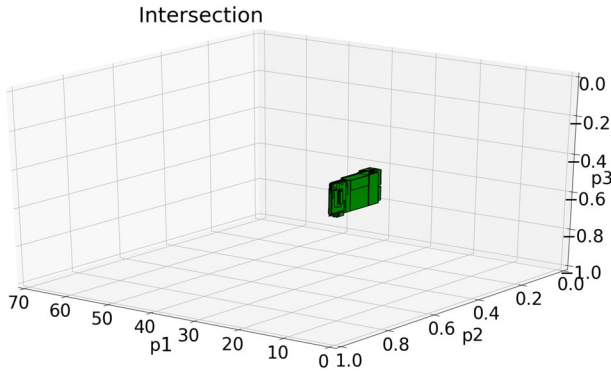
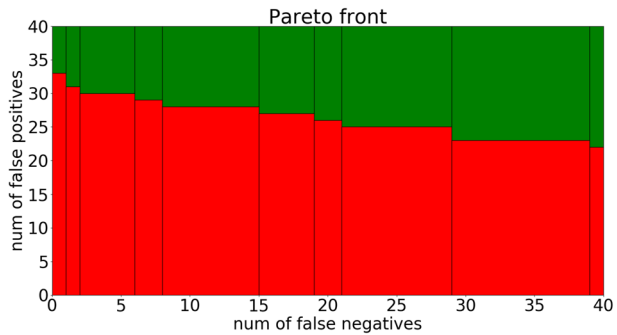


Fig. 16 Solution set of the parametric space $\text{DomInter}^\sigma(\Psi^{ch_1}, \mathcal{S}_{221}, f_+ = 1, f_- = 0)$ for ECG221 ($V_\delta = 0.01\%$)

Table 2 Execution time (seconds) and number of f_-/f_+ of the solution set of the parametric space for different ECGs

ECG n°	f_-	f_+	$V_\delta = 1\%$	$V_\delta = 0.1\%$	$V_\delta = 0.01\%$
221	0	1	55.05	185.67	861.70
123	1	0	90.29	393.70	2265.52
100	0	33	507.60	3891.94	12780.00

Fig. 17 Pareto front for ECG100 using Eq. (10)



The Cython code is around 3% faster than pure Python: the evaluation of the STL expressions, which is the most time consuming part of the mining method, was already externalised to StEval in previous versions of ParetoLib. Additionally, ParetoLib 2.0 supports the parallel execution of the BDMJ20 method in multicore CPU’s, while ParetoLib 1.0 did not. The experiments are run in a Intel(R) Core(TM) i5-3570K CPU @3.40GHz 16GB, Ubuntu 22.04 and Python 3.10.

Equation (10) is also evaluated for labelling pulses on ECG 100 and ECG 123 with different success. We denote the singleton signal sets containing the labelled ECG 100 and ECG 123 signals as \mathcal{S}_{100} and \mathcal{S}_{123} respectively. Table 2 compares the number of mismatches, either false positives or false negatives, between the original labelling and the one produced by Eq. (10). For the particular case of ECG 100, our solution provides 33 false positives. According to the Pareto front (corresponding to the upset $\mathcal{P}(\Psi^{ch_1}, \mathcal{S}_{100})$)

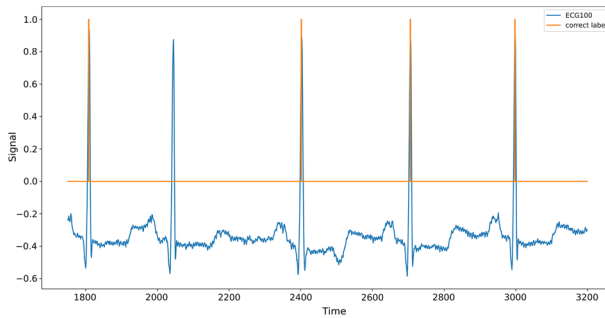


Fig. 18 Excerpt from ECG100

in Fig. 17, it is impossible to obtain a better solution set using the current specification. This issue does not come from a wrong definition of peak, but from pulses that are separated further than usual. See the first pulse after 2000 time units in Fig. 18. It is called an Atrial Premature Beat which should not be considered a normal peak. Equation 12 enriches Eq. (10) with explicit information about the distance between peaks, which improves the accuracy of our peak detector and reduces the number of errors to 3 false positives and 1 false negative. *peak* is an alias for Eq. (10).

$$\Psi_{(p_1, p_2, p_3, p_4)}^{ch_3}(\mathbf{x})(t) := peak(p_1, p_2, p_3) \wedge F_{[0, p_4]} peak(p_1, p_2, p_3) \quad (12)$$

6 Related work

Similar tools for offline monitoring of cyber-physical systems are AMT 2.0 [18], S-Taliro [25] and Breach [26]. AMT 2.0 is a Java tool while S-Taliro and Breach are MATLAB/Simulink toolboxes. AMT 2.0 analyses input traces with extended Signal Temporal Logic (xSTL), which combines STL and Timed Regular Expressions (TRE). On the other hand, S-Taliro and Breach include an explicit model of the cyber-physical system for simulating traces. S-Taliro is specialised in falsification of temporal logic properties by finding trajectories with minimum robustness. Breach allows the exhaustive inspection of the cyber-physical model by systematically varying configuration parameters. Next, py-stl [27, 28] implements a similar approach to our BBMJ19 method for mining parametric STL equations. The authors of py-stl apply it to compute a family of distance metrics for a set of monotonic specifications that are mined from time-series learning [29]. Finally, MiniPaSTeL [30] implements the BMNN23 method for mining STL parameters [7]. It uses RTAMT [31] as monitoring tool.

However, these tools include neither the quantitative operators nor a full support for the specification mining methods presented in this paper. Additionally, ParetoLib 2.0 has an adapter for externalising the run of non-parametric STL queries in AMT 2.0 and is also prepared for connecting with the Matlab environment. Similarly to py-stl, ParetoLib 2.0 supports the comparison of the inferred validity domains for mined PSTL specifications. Our tool can compute the Hausdorff distances among a set of validity domains and return the characteristic point for each of them.

Recently, tools for online monitoring of systems that cannot be statically analysed have appeared. RTAMT [31] supports the online diagnosis with STL and IA-STL, an *interface-aware* extension for defining interface properties about input/output signals of a cyber-physical systems. Finally, RTLola [32] (previously named StreamLab) and TeSSLa [33] complete the state of the art. These tools are focused on evaluating real-time streams instead of signals.

7 Conclusion

This paper introduces the new features of ParetoLib 2.0, a Python tool for the evaluation and parameter synthesis of Signal Temporal Logic specifications (STL). The main changes w.r.t. the previous version of the tool consist of (1) A graphical user interface that simplifies the usage to the end users, (2) The support of additional quantitative operators that involve *counting of events* (ϵ -count), *trends* (derivatives), or *accumulations* (integrals), and (3) The implementation of new mining methods. Besides, we have optimised the performance of the library for mining the parameter valuations of parametric STL specifications by completing the multi core support and compiling the kernel modules into C code. The compilation of the internal Python modules into C code is transparent to the end users as the transformation is automatically executed during the installation of the Python library. Finally, we introduce the Hausdorff distance in order to compare the inference of validity domains of PSTL specifications.

As future work, we propose to include more options in the graphical user interface that are already available in ParetoLib as command-line options (e.g., the selection of parallel computations or the STL engine). On the other hand, we want to include other types of interpolation beyond the constant pairwise (e.g. linear). We plan to add new logic operators such as the *probability* operator $\mathbf{Pr}_{\sim, \lambda} \varphi$, where $\sim \in \{<, \leq, \geq, >\}$ and $\lambda \in [0, 1]$. Finally, we will implement a natural language processor for writing STL specifications in natural language.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Data availability The code and datasets for comparing Parma Polyhedra Library [10] and ParetoLib in Sect. 2 is publicly available in <https://zenodo.org/doi/10.5281/zenodo.10156861>. The Python scripts and the subset of electrocardiograms (ECGs) from the MIT-BIH Arrhythmia Database of Physionet [23, 24] that are used in our case study are located in the `doc/examples/OracleEpsSTLe` folder of the ParetoLib tool [6]. The ParetoLib tool is publicly available at https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/tempo/multi_dimensional_search. The MIT-BIH Arrhythmia Database of Physionet is publicly available at <https://physionet.org/content/mitdb/1.0.0/>.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.


References

1. Maler O, Nickovic D (2004) Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) Formal techniques, modelling and analysis of timed and fault-tolerant systems, joint international conferences on formal modelling and analysis of timed systems, FORMATS 2004 and formal techniques in real-time and fault-tolerant systems, ftrftt 2004, september 22–24, 2004, proceedings. Lecture notes in computer science, vol. 3253, pp. 152–166. Springer, Grenoble, France. https://doi.org/10.1007/978-3-540-30206-3_12
2. Asarin E, Donzé A, Maler O, Nickovic D (2011) Parametric identification of temporal properties. In: Khurshid, S., Sen, K. (eds.) Runtime verification - second international conference, RV 2011, September 27–30, 2011, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7186, pp. 147–160. Springer, San Francisco, CA, USA. https://doi.org/10.1007/978-3-642-29860-8_12
3. Bakhirkin A, Basset N (2019) Specification and efficient monitoring beyond STL. In: Vojnar, T., Zhang, L. (eds.) Tools and algorithms for the construction and analysis of systems - 25th international conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, April 6–11, 2019, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11428, pp. 79–97. Springer, Prague, Czech Republic. https://doi.org/10.1007/978-3-030-17465-1_5
4. Bakhirkin A, Basset N, Maler O, Jarabo JR (2019) Paretolib: A python library for parameter synthesis. In: André, É., Stoelinga, M. (eds.) Formal modeling and analysis of timed systems - 17th international conference, FORMATS 2019, August 27–29, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11750, pp. 114–120. Springer, Amsterdam, The Netherlands. https://doi.org/10.1007/978-3-030-29662-9_7
5. Basset N, Dang T, Mambakam A, Jarabo JR (2020) Learning specifications for labelled patterns. In: Bertrand, N., Jansen, N. (eds.) Formal modeling and analysis of timed systems - 18th international conference, FORMATS 2020, September 1–3, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12288, pp. 76–93. Springer, Vienna, Austria. https://doi.org/10.1007/978-3-030-57628-8_5
6. Mambakam A, Jarabo JR (2022) ParetoLib, 2.X, VERIMAG Git Repository. https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/tempo/multidimensional_search
7. Aguilar EA, Bartocci E, Mateis C, Nesterini E, Nickovic D (2023) Mining specification parameters for multi-class classification. In: Katsaros, P., Renzi, L. (eds.) Runtime verification - 23rd international conference, RV 2023, Thessaloniki, Greece, October 3–6, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14245, pp. 86–105. Springer, ????. https://doi.org/10.1007/978-3-031-44267-4_5
8. Bakhirkin A, Ferrère T, Henzinger TA, Nickovic D (2018) The first-order logic of signals: keynote. In: Brandenburg, B.B., Sankaranarayanan, S. (eds.) Proceedings of the international conference on embedded software, EMSOFT 2018, September 30 - October 5, 2018, pp. 1. IEEE, Torino, Italy. <https://doi.org/10.1109/EMSOFT.2018.8537203>
9. Brim L, Dluhos P, Safránek D, Vejpustek T (2014) Stl*: Extending signal temporal logic with signal-value freezing operator. Inf Comput 236:52–67. <https://doi.org/10.1016/j.ic.2014.01.012>
10. Bagnara R, Hill PM, Zaffanella E (2008) The Parma polyhedra library: toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. Sci Comput Program 72(1–2):3–21. <https://doi.org/10.1016/j.SCICO.2007.08.001>
11. Requeno JI (2022) Detection of smart grid integrity attacks using signal temporal logic. CoRR [abs/2209.06722](https://arxiv.org/abs/2209.06722) [arXiv:2209.06722](https://arxiv.org/abs/2209.06722). <https://doi.org/10.48550/arXiv.2209.06722>
12. Donzé A, Maler O (2010) Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) Formal modeling and analysis of timed systems - 8th international conference, FORMATS 2010, Klosterneuburg, Austria, September 8–10, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6246, pp. 92–106. Springer, ????. https://doi.org/10.1007/978-3-642-15297-9_9
13. Buyukkocak AT, Aksaray D, Yazicioglu Y (2021) Control synthesis using signal temporal logic specifications with integral and derivative predicates. In: 2021 American control conference, ACC 2021, May 25–28, 2021, pp. 4873–4878. IEEE, New Orleans, LA, USA. <https://doi.org/10.23919/ACC50511.2021.9482651>
14. Lemire D (2006) Streaming maximum–minimum filter using no more than three comparisons per element. Nordic J Comput 13(4)
15. Donzé A, Ferrère T, Maler O (2013) Efficient robust monitoring for STL. In: Sharygina, N., Veith, H. (eds.) Computer aided verification - 25th international conference, CAV 2013, July 13–19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8044, pp. 264–279. Springer, Saint Petersburg, Russia. https://doi.org/10.1007/978-3-642-39799-8_19
16. Kolmogorov AN, Tikhomirov VM (1959) ϵ -entropy and ϵ -capacity of sets in function spaces. Uspekhi Matematicheskikh Nauk 14(86):386
17. Bakhirkin A, Mambakam A, Jarabo JR (2022) StlEval, 2.X, VERIMAG Git Repository. <https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/tempo/StlEval>

18. Nickovic D, Lebeltel O, Maler O, Ferrère T, Ulus D (2020) AMT 2.0: qualitative and quantitative trace analysis with extended signal temporal logic. *Int J Softw Tools Technol Transf* 22(6):741–758. <https://doi.org/10.1007/s10009-020-00582-z>
19. Bakhrkin A, Basset N, Maler O, Requeno JI (2019) Learning pareto front from membership queries. working paper or preprint. <https://hal.archives-ouvertes.fr/hal-02125140>
20. Behnel S, Bradshaw R, Citro C, Dalcin L, Seljebotn DS, Smith K (2010) Cython: The best of both worlds. *Comput Sci Eng* 13(2):31–39
21. Maler O (2017) Learning monotone partitions of partially-ordered domains (Work in Progress). working paper or preprint. <https://hal.archives-ouvertes.fr/hal-01556243>
22. Bakhrkin A, Basset N, Maler O, Requeno JI (2019) Learning pareto front from membership queries. working paper or preprint . <https://hal.archives-ouvertes.fr/hal-02125140>
23. Goldberger AL, Amaral LA, Glass L, Hausdorff JM, Ivanov PC, Mark RG, Mietus JE, Moody GB, Peng C-K, Stanley HE (2000) Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. *Circulation* 101(23):215–220
24. Moody GB, Mark RG (2001) The impact of the MIT-BIH arrhythmia database. *IEEE Eng Med Biol Mag* 20(3):45–50
25. Annpureddy Y, Liu C, Fainekos G, Sankaranarayanan S (2011) S-talro: a tool for temporal logic falsification for hybrid systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) Tools and algorithms for the construction and analysis of systems - 17th international conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, March 26–April 3, 2011. Proceedings. *Lecture Notes in Computer Science*, vol. 6605, pp. 254–257. Springer, Saarbrücken, Germany. https://doi.org/10.1007/978-3-642-19835-9_21
26. Donzé A (2010) Breach, A toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P.B. (eds.) Computer aided verification, 22nd international conference, CAV 2010, July 15–19, 2010. Proceedings. *Lecture Notes in Computer Science*, vol. 6174, pp. 167–170. Springer, Edinburgh, UK . https://doi.org/10.1007/978-3-642-14295-6_17
27. Vazquez-Chanlatte M (2017) Py-signal-temporal-logic, 1.0, GitHub. <https://github.com/mvcisback/py-signal-temporal-logic>
28. Vazquez-Chanlatte M, Deshmukh JV, Jin X, Seshia SA (2017) Logical clustering and learning for time-series data. In: Majumdar, R., Kuncak, V. (eds.) Computer aided verification - 29th international conference, CAV 2017, July 24–28, 2017. Proceedings, Part I. *Lecture Notes in Computer Science*, vol. 10426, pp. 305–325. Springer, Heidelberg, Germany. https://doi.org/10.1007/978-3-319-63387-9_15
29. Vazquez-Chanlatte M, Ghosh S, Deshmukh JV, Sangiovanni-Vincentelli AL, Seshia SA (2018) Time-series learning using monotonic logical properties. In: Colombo, C., Leucker, M. (eds.) Runtime verification - 18th international conference, RV 2018, Limassol, Cyprus, November 10–13, 2018. Proceedings. *Lecture Notes in Computer Science*, vol. 11237, pp. 389–405. Springer, Cham. https://doi.org/10.1007/978-3-030-03769-7_22
30. Nesterini E (2023) MiniPaSTeL, GitHub Repository. <https://github.com/eleonoranesterini/MiniPaSTeL>
31. Nickovic D, Yamaguchi T (2020) RTAMT: online robustness monitors from STL. In: Hung, D.V., Sokolsky, O. (eds.) Automated technology for verification and analysis - 18th international symposium, ATVA 2020, October 19–23, 2020. Proceedings. *Lecture Notes in Computer Science*, vol. 12302, pp 564–571. Springer, Hanoi, Vietnam. https://doi.org/10.1007/978-3-030-59152-6_34
32. Faymonville P, Finkbeiner B, Schledjewski M, Schwenger M, Stenger M, Tentrup L, Torfah H (2019) Streamlab: Stream-based monitoring of cyber-physical systems. In: Dillig, I., Tasiran, S. (eds.) Computer aided verification - 31st international conference, CAV 2019, July 15–18, 2019. Proceedings, Part I. *Lecture Notes in Computer Science*, vol. 11561, pp 421–431. Springer, New York City, NY, USA . https://doi.org/10.1007/978-3-030-25540-4_24
33. Leucker M, Sánchez C, Scheffel T, Schmitz M, Schramm A (2018) Tessa: runtime verification of non-synchronized real-time streams. In: Haddad, H.M., Wainwright, R.L., Chbeir, R. (eds.) Proceedings of the 33rd annual ACM symposium on applied computing, SAC 2018, April 09–13, 2018, pp 1925–1933. ACM, Pau, France., <https://doi.org/10.1145/3167132.3167338>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

Akshay Mambakam¹ · **José Ignacio Requeno Jarabo**²  · **Alexey Bakhirkin**^{1,2} · **Nicolas Basset**¹ · **Thao Dang**¹

✉ José Ignacio Requeno Jarabo
jrequeno@ucm.es

Akshay Mambakam
akshay.mambakam@univ-grenoble-alpes.fr

Alexey Bakhirkin
abakhirkin@gmail.com

Nicolas Basset
nicolas.basset1@univ-grenoble-alpes.fr

Thao Dang
thao.dang@univ-grenoble-alpes.fr

¹ VERIMAG/CNRS, University Grenoble Alpes, 700 Av. Centrale, 38400 Saint-Martin-d'Hères, France

² Department of Information Systems and Computing, Complutense University of Madrid, Calle del Prof. José García Santesmases, 9, 28040 Madrid, Spain

Terms and Conditions

Springer Nature journal content, brought to you courtesy of Springer Nature Customer Service Center GmbH (“Springer Nature”).

Springer Nature supports a reasonable amount of sharing of research papers by authors, subscribers and authorised users (“Users”), for small-scale personal, non-commercial use provided that all copyright, trade and service marks and other proprietary notices are maintained. By accessing, sharing, receiving or otherwise using the Springer Nature journal content you agree to these terms of use (“Terms”). For these purposes, Springer Nature considers academic use (by researchers and students) to be non-commercial.

These Terms are supplementary and will apply in addition to any applicable website terms and conditions, a relevant site licence or a personal subscription. These Terms will prevail over any conflict or ambiguity with regards to the relevant terms, a site licence or a personal subscription (to the extent of the conflict or ambiguity only). For Creative Commons-licensed articles, the terms of the Creative Commons license used will apply.

We collect and use personal data to provide access to the Springer Nature journal content. We may also use these personal data internally within ResearchGate and Springer Nature and as agreed share it, in an anonymised way, for purposes of tracking, analysis and reporting. We will not otherwise disclose your personal data outside the ResearchGate or the Springer Nature group of companies unless we have your permission as detailed in the Privacy Policy.

While Users may use the Springer Nature journal content for small scale, personal non-commercial use, it is important to note that Users may not:

1. use such content for the purpose of providing other users with access on a regular or large scale basis or as a means to circumvent access control;
2. use such content where to do so would be considered a criminal or statutory offence in any jurisdiction, or gives rise to civil liability, or is otherwise unlawful;
3. falsely or misleadingly imply or suggest endorsement, approval, sponsorship, or association unless explicitly agreed to by Springer Nature in writing;
4. use bots or other automated methods to access the content or redirect messages
5. override any security feature or exclusionary protocol; or
6. share the content in order to create substitute for Springer Nature products or services or a systematic database of Springer Nature journal content.

In line with the restriction against commercial use, Springer Nature does not permit the creation of a product or service that creates revenue, royalties, rent or income from our content or its inclusion as part of a paid for service or for other commercial gain. Springer Nature journal content cannot be used for inter-library loans and librarians may not upload Springer Nature journal content on a large scale into their, or any other, institutional repository.

These terms of use are reviewed regularly and may be amended at any time. Springer Nature is not obligated to publish any information or content on this website and may remove it or features or functionality at our sole discretion, at any time with or without notice. Springer Nature may revoke this licence to you at any time and remove access to any copies of the Springer Nature journal content which have been saved.

To the fullest extent permitted by law, Springer Nature makes no warranties, representations or guarantees to Users, either express or implied with respect to the Springer nature journal content and all parties disclaim and waive any implied warranties or warranties imposed by law, including merchantability or fitness for any particular purpose.

Please note that these rights do not automatically extend to content, data or other material published by Springer Nature that may be licensed from third parties.

If you would like to use or distribute our Springer Nature journal content to a wider audience or on a regular basis or in any other manner not expressly permitted by these Terms, please contact Springer Nature at

onlineservice@springernature.com