Modular and Online Monitoring of Temporal Logic Specification with Integral and Filter (Extended Version)

Simone Silvetti[1]✉[0000−0001−8048−9317], Michele Loreti[2][0000−0003−3061−863X], and Laura Nenzi[1][0000−0003−2263−9342]

[1] University of Trieste, Trieste, Italy
simone.silvetti@dia.units.it, lnenzi@units.com
[2] University of Camerino, Camerino, Italy
michele.loreti@unicam.it

Abstract In a recent work, Bakhirkin and Basset [3] proposed a new specification language that extends STL. Their logic overcomes the syntactic restrictions of STL, enabling the production and manipulation of real-valued output signals and the expression of properties that have typically been described using other logics, such as STL*. In this contribution, we extend this specification language in three directions. First, we introduce a novel integral operator over sliding windows, allowing the specification of cumulative properties, for example, asserting that the integral of a signal over a time interval remains within a given threshold. Second, we introduce a filtering operator for the sliding window operator, enabling us to restrict the scope of aggregation to signal segments that satisfy the filtering condition. Third, we develop an efficient online monitoring algorithm for the extended logic. Finally, we test the logic on two case studies: an artificial pancreas controller and a monitoring of outdoor weather events.

Keywords: Signal Temporal Logic · Cumulative Temporal Properties · Runtime Verification · Monitoring

1 Introduction

As digitalization advances, smart cities and Industry 4.0 environments rely on sensors that generate massive amounts of data daily, capturing information on traffic, pollution, machine status, and energy usage [19]. These settings demand systems capable of real-time analysis and prompt reaction to critical changes. For instance, a smart insulin pump must continuously adapt insulin delivery based on fluctuating glucose levels, while an autonomous traffic management system dynamically adjusts signal timings to alleviate congestion. Ensuring the reliability and responsiveness of such systems requires formal specifications and verification methods that operate at runtime.

To this end, *temporal logics* such as Metric Interval Temporal Logic (MITL) and Signal Temporal Logic (STL) [20] have become standard tools for specifying temporal properties over real-valued signals. STL, in particular, has gained popularity due to its ability to express time-bounded properties over continuous signals and its efficient monitoring algorithms for both Boolean and quantitative (robustness) semantics [26,11].

However, despite these strengths, STL has significant expressiveness limitations. For example, it cannot naturally express cumulative temporal properties such as "the level of glucose is above the hyperglycemia threshold for more than 1 hour", nor can it easily detect oscillatory behaviors or stabilization patterns. To overcome these limitations, several extended formalisms have been proposed. Signal Convolution Logic (SCL) [24] introduces convolution-based operators to express cumulative properties, while STL* [8] incorporates freeze quantifiers for referencing signal values at specific time points, allowing for expressing oscillatory or stabilizing behavior. A key challenge of these extensions is that their increased expressiveness often leads to more complex monitoring algorithms.

A notable recent contribution by Bakhirkin and Basset [3] introduces a specification language that extends STL by removing its syntactic constraints, allowing each formula to produce a real-valued output signal and combine them in a pointwise way with arithmetic operations, comparisons, etc. A key point of this work is that the extended logic preserves the linear-time (offline) monitoring complexity of STL while enabling more expressive specifications, such as stabilization and oscillatory behavior, typically described with STL*.

In this work, we extend the approach of Bakhirkin and Basset [3] in three directions. First, we introduce an *integral operator* over sliding windows, enabling the specification of cumulative properties, such as asserting that the integral of a signal over a time interval remains within a threshold. This addition supports reasoning about system behaviors that rely on accumulation or averaging, common in domains like energy systems and biomedical monitoring. Second, we introduce a *filtering operator* for the sliding windows operator, allowing us to reduce the scope of the aggregation to the signal satisfying the filtering condition. Third, we develop an efficient *online monitoring algorithm* for the extended logic. Unlike offline techniques that process entire traces post-execution, our algorithm processes signals incrementally as they arrive, making it suitable for real-time monitoring. Furthermore, we provide an efficient implementation for both piecewise-constant and piecewise-linear signals, proving that the property of finite variability is preserved. Our contribution thus bridges the gap between expressiveness and efficiency in runtime monitoring, enabling the specification and real-time evaluation of a richer class of temporal properties, essential for the dependable operation of modern Cyber-Physical Systems (CPS) applications.

Running examples. We consider the following two representative examples to illustrate the types of temporal properties that motivate our work.

P1) *Weather Events* – The first example considers a network of Internet of Things (IoT) sensors capable of measuring temperature and CO_2 emissions [25], among

other variables, to detect emerging issues during extreme weather events. A typical scenario, associated with a high risk of fire accident, arises when CO_2 and temperature sensors report average readings exceeding 400 ppm and 60°C over a one-hour interval. Formally, this can be written as:

$$\varphi_W(T, \theta_1, \theta_2) := \left(\frac{1}{T}\text{On}_{[0,T]}\ \text{Int}\ CO_2 > \theta_1\right) \wedge \left(\frac{1}{T}\text{On}_{[0,T]}\ \text{Int}\ temp > \theta_2\right)$$

with $\theta_1 = 400$, $\theta_2 = 60$, and $T = 3600$ sec. Note that this property cannot be expressed in SCL, because its cumulative operators act only on Boolean signals (e.g., $\text{CO}_2(t) > \theta_1$), rather than directly on the numerical value of $\text{CO}_2(t)$.

P2) *Artificial Pancreas* – It is a closed-loop system for treating type 1 diabetes, where a continuous glucose monitor (CGM) tracks blood sugar levels and a software-controlled insulin pump delivers insulin automatically.

To assess glycemic (concentration of glucose in the blood) control, standard CGM-derived metrics [7] are: the Time Above Range (TAR), representing glucose level exceeding 180 mg/dL (i.e., $\varphi_{AR} := G \geq 180$), and Time Below Range (TBR) where the glucose level is below 70 mg/dL (i.e., $\varphi_{BR} := G \leq 70$). Beyond these, the mean glucose level over a certain interval of time is a core indicator of overall glycemic status and can be represented with the following formulas:

$$\psi_{TAR} := \tfrac{1}{3h}\text{On}_{3h}\text{Int}\,\varphi_{AR}, \quad \psi_{TBR} := \tfrac{1}{3h}\text{On}_{3h}\text{Int}\,\varphi_{BR}$$

Furthermore, the conditional mean glucose values, restricted to periods of hyperglycemia and hypoglycemia, provide insight into the average glycemic burden during clinically significant excursions. These formulas can be formally written as:

$$\psi_{\mu GAR} := \tfrac{1}{3h}\text{On}_{3h}\text{Int}\,(G|\varphi_{AR}), \quad \psi_{\mu GBR} := \tfrac{1}{3h}\text{On}_{3h}\text{Int}\,(G|\varphi_{BR})$$

The utility of these conditional averages has been highlighted in recent CGM-based studies for their role in understanding the physiological impact of glucose extremes and tailoring individualized treatment strategies [15,12]. The rationale behind using a conditional glucose mean is straightforward: while TAR and TBR are important indicators for identifying hyperglycemia and hypoglycemia, respectively, they do not capture the severity of these conditions, something the proposed metrics characterize more effectively.

These examples underscore the need for a specification framework that supports both expressive and efficiently monitorable properties, particularly in safety-critical and time-sensitive domains. For other interesting properties regarding stabilization or oscillatory events, please refer to [3].

Related Work. Several works were introduced to extend the expressivity of STL, e.g., with existential quantification [4], or freeze quantification as in STL* [8]. With respect to the integral operator, Akazaki et al. [1] extend the syntax of STL with averaged temporal operators, specified time interval. The new operators require the definition of two types of robustness (positive and negative), which

results in the loss of the correctness property. In [9], the authors introduce the integral operator but restricted to predicates; in [24], the authors presented the Signal Convolution Logic (SCL). The logic has a convolution operator that allows integration on Boolean constraints (i.e., it can specify the percentage of time a certain property is satisfied), but it does not permit describing properties as P1 and P2, averaging, and/or filtering directly the values of the signal.

An important challenge with these extensions is that the increased expressiveness of the new operators comes at the cost of more complex monitoring algorithms. The approach proposed by Bakhirkin and Basset [3] addresses this limitation by defining a specification language that is more expressive than STL, yet still efficient for offline monitoring. The core idea is to remove syntactic constraints from the temporal logic language. This enables the generation and manipulation of real-valued output signals, allowing for the direct expression of properties such as stabilization, local maxima, and linear growth. The logic supports aggregation using min and max functions, but does not include an integral and a filtering operator.

Online monitoring of Signal Temporal Logic (STL) has been explored to enable real-time verification of system behaviors during execution [26,11]. [11] introduces an approach to monitor STL robustness online by incrementally computing signal properties as new data arrives, considering an interval semantics. The tool RTAMT [26] supports online monitoring with efficient implementations. Despite these advances, full support for STL extensions in an online context remains an open challenge.

An interesting way to design online monitoring algorithms is to use transducers. The first contribution on this topic is [22], the paper introduces the temporal testers, a class of transducer automata that emit a Boolean signal at each step, indicating whether a temporal logic formula holds at that point in time. [21] presents a modular and algebraically founded approach to online monitoring of Metric Temporal Logic (MTL) formulas over continuous-time piecewise-constant signals, supporting both qualitative and quantitative semantics and managing uncertainty through compositional and deterministic signal transducers.

In the domain of stream-based monitoring, RTLola [14] supports sliding window aggregation. However, unlike our approach, it evaluates aggregates at a fixed, user-defined rate (e.g., integrating a signal every 10 ms). TeSSLa [18], on the other hand, provides a set of primitive operations that, when appropriately composed, can approximate the behavior of an integral operator. Nevertheless, it does not natively account for the *duration* a signal maintains a particular value during evaluation, limiting its ability to capture time-weighted integrals accurately.

2 Preliminaries

2.1 Dual Numbers.

Dual numbers extend the real numbers $\mathbb{R}$ to the algebra $\mathbb{R}_\varepsilon = \{a + b\varepsilon \mid a, b \in \mathbb{R}\}$, where ε is a symbol satisfying the identity $\varepsilon^2 = 0$. Intuitively, ε represents an

infinitesimal value, and $a + b\varepsilon$ for $b \neq 0$ is a number approaching a with slope b. Dual numbers, introduced in [10], have been applied to several topics ranging from geometry and mechanics to automatic differentiation. In this contribution, we use them to extend the concept of minimum and maximum to functions with discontinuities [3]. Consider indeed the following functions:

$$s_1(t) = \begin{cases} 1+t, & t \in [0.5, 1), \\ 1, & t \geq 1, \end{cases} \qquad s_2(t) = \begin{cases} 1+2t, & t \in [0.5, 1), \\ 2, & t \geq 1, \end{cases}$$

If we restrict to $\mathbb{R}$, the maximum of s_1 in $[0, 2]$ is not defined because there always exists another point in the left neighbourhood of 1, with a higher value. Using dual numbers, instead, we define the maximum as $\max_{[0,2]} s_1 = \max(s_1(0), s_1(1-\varepsilon), s_1(2)) = 2 - \varepsilon$.

Dual numbers are particularly useful for computing relations among functions. Consider, for example, the requirement that $\forall t \in [0, 2]$, $s_1(t) < s_2(t)$. A standard approach to verify such a requirement is to verify if $\max_{[0,2]} s_1 < \min_{[0,2]} s_2$ which is true in $\mathbb{R}_\varepsilon$ considering that $\max_{[0,2]} s_1 = 2 - \varepsilon < 2 = \min_{[0,2]} s_2$. On the contrary, in $\mathbb{R}$ we cannot achieve the same with inf and sup operators.

2.2 Signal and trace.

Definition 1 (Symbolic State). *A symbolic state (or f-interval) $v = (I, f)$ is a pair where $I = [a, b) \subseteq \mathbb{R}$ is an interval and $f : \mathbb{R} \to \mathbb{R}$ a continuous function.*

We say that two symbolic states $v_1 = (I_1, f_1)$ and $v_2 = (I_2, f_2)$ are *adjacent* if the corresponding intervals are contiguous, i.e., $\sup(I_1) = \min(I_2)$ and that $(v_1; v_2)$ is a *refinement* of $v = (I, f)$ if v_1 and v_2 are adjacent, $I = I_1 \cup I_2$ and $f_1|_{I_1} = f|_{I_1}$ and $f_2|_{I_2} = f|_{I_2}$. Moreover, we say that v is *resolved* if it does not allow a refinement $(u; w)$ such that $f_u \neq f_w$.

We associate to any (adjacent) succession of symbolic states $v = (v_1, v_2, \dots)$ a function $f_v : \mathbb{R}_{\geq 0} \to \mathbb{R} \cup \{\texttt{NaN}\}$ such that if $v_i = (I_i, f_i)$ and $t \in I_i$ then $f_v(t) = f_i(t)$ while $f_v(t) = \texttt{NaN}$ if for any $i \in \mathbb{N}$ such that $t \notin I_i$, meaning that f_v assumes an undefined value represented by symbol `NaN`. As an example, consider succession $(([0, 3), x), ([3, 10), 6 - x))$. It assumes defined values in $[0, 10)$ and the undefined value in $[10, +\infty)$. We impose that any function with `NaN` as one of its inputs produces `NaN` as output independently of the other input values. So for example $\texttt{NaN} \cdot 0 + 5 = \texttt{NaN}$.

Definition 2 (Signal). *A signal is a function $s : \mathbb{T} \subseteq \mathbb{R}_{\geq 0} \to \mathbb{R}$ that can be represented as a succession of symbolic states, meaning that there exists $v \in \mathcal{A}$ such that $s = f_v$.*

Remark 1. Let's consider two finite sequences of (adjacent) symbolic states, u and w. It is always possible to refine these sequences into new ones, $u' = \{(I_i, f_i)\}_{i \leq n}$ and $w' = \{(J_i, g_i)\}_{i \leq n}$, such that u' is equivalent to u and w' is

equivalent to w, where equivalent means that they represent the same function. Moreover, this refinement can be done in such a way that $\forall i \leq n$, $I_i = J_i$, meaning the intervals are aligned (i.e., overlapping exactly) and $n \leq |u| + |v|$.

Definition 3 (Finite Variability Property). *A signal satisfies the finite variability property (fvp) if each of its restrictions to a finite length interval admits a representation as a finite succession of resolved symbolic states.*

The finite variability property was initially introduced for Boolean signals in [2] and later formalized for piecewise-constant signals in [20]. Different from the previous work, our definition is not saying that the signal assumes a finite number of states in a finite time window, but more generally, we are saying that the signals admit finite mathematical representations (as a continuous function) in a finite time window. This broader view motivates our use of the term symbolic state. In this sense, our definition aligns with the one proposed in [4] for piecewise-linear functions.

Definition 4 (Trace). *Let $n \in \mathbb{N}$, we define trace, or multi-valued signal, any function $\boldsymbol{s} : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ such that $\boldsymbol{s} = (s_1, \ldots, s_n)$ and $s_i : \mathbb{R}_{\geq 0} \to \mathbb{R}$ are signals.*

As an example, let's consider the first running example (P1) where the trace $\boldsymbol{s}(t) = (temp(t), CO_2(t))$ is composed of two signals describing respectively the temperature and the level of CO_2.

3 Syntax and Semantics

The syntax of our language is given by

$$\begin{aligned}
\varphi &:= X_i \mid \chi \mid op(\varphi_1, \ldots, \varphi_n) \mid \mathrm{On}_T(\psi) \mid \psi \mathrm{U}_T^d \chi \mid \varphi \downarrow \mathrm{U}_T^d \chi \\
\psi &:= \mathrm{Min}\, \varphi \mid \mathrm{Max}\, \varphi \mid \mathrm{Int}\, \varphi \\
\chi &:= \top \mid \bot \mid \varphi \bowtie 0 \mid \neg \chi \mid \chi \wedge \chi
\end{aligned}$$

where X_i is a variable indexed with $i \in \mathbb{N}$, op is a generic n-ary pointwise operator, $n \in \mathbb{N}$, On_T is the sliding window operator (or On-operator) with $T \subset \mathbb{R}_{\geq 0}$ a closed interval, $d \in \mathbb{R}$, U_T^d and $\downarrow \mathrm{U}_T^d$ are respectively the find-first aggregation and the find-first pointwise until. Min, Max, and Int are, respectively, the minimum, maximum, and integral operators that serve as aggregation functions for the sliding window operator. Negation $\neg$ and conjunction $\wedge$ are the standard Boolean connectives, and $\bowtie \in \{>, \geq, <, \leq\}$. In addition, we can derive disjunction operator $\vee$ as usual, and the temporal operator of STL[3], as follows:

$$\chi_1 U_T \chi_2 := (\mathrm{Min}\ \chi_1)\mathrm{U}_T^0 \chi_2 \qquad G_T(\chi) := \mathrm{On}_T \mathrm{Min}\ \chi \qquad F_T(\chi) := \mathrm{On}_T \mathrm{Max}\ \chi$$

We also introduce the filtered version of the On-operator, defined as

$$\mathrm{On}_T(\mathrm{Agg}\, \varphi | \chi) := \mathrm{On}_T(\mathrm{Agg}(\varphi \text{ if } \chi \text{ else } \texttt{NaN})) \tag{1}$$

[3] For a deeper discussion on this equivalence please refer to [3]

where (ψ if χ else NaN) is a pointwise operator useful to limit the scope of the sliding operator to the region of the time domain where a filtering condition (i.e., χ) holds, and Agg is a placeholder for Min, Max and Int.

The semantics of our logic is evaluated pointwise at each time.

Definition 5 (Semantics). *We introduce the semantics function* $[\![\cdot]\!] : \mathcal{T} \times \mathbb{R}_{\geq 0} \times \mathcal{L} \to \mathbb{R}$*, where* $\mathcal{T}$ *is the trace domain, and* $\mathcal{L}$ *the logic, recursively as follows:*

$$
\begin{aligned}
[\![\boldsymbol{s}, t, X_i]\!] &:= s_i(t) \\
[\![\boldsymbol{s}, t, op(\varphi_1, \ldots, \varphi_n)]\!] &:= op([\![\boldsymbol{s}, t, \varphi_1]\!], \ldots, [\![\boldsymbol{s}, t, \varphi_n]\!]) \\
[\![\boldsymbol{s}, t, On_T(Agg\,\varphi)]\!] &:= Agg_{\tau \in t+T}[\![\boldsymbol{s}, \tau, \varphi]\!] \\
[\![\boldsymbol{s}, t, (Agg\,\varphi)\,U_T^d\chi]\!] &:= \begin{cases} Agg_{t \in F(t,T,\chi)}[\![\boldsymbol{s}, \tau, \varphi]\!] \text{ if } F(t,T,\chi) \neq \emptyset \\ d \text{ otherwise} \end{cases} \\
[\![\boldsymbol{s}, t, \varphi \downarrow U_T^d\chi]\!] &:= \begin{cases} [\![\boldsymbol{s}, \tau, \varphi]\!] \text{ if } F(t,T,\chi) = [t,\tau] \neq \emptyset \\ d \text{ otherwise} \end{cases} \\
[\![\boldsymbol{s}, t, \top]\!] &:= 1 \\
[\![\boldsymbol{s}, t, \bot]\!] &:= 0 \\
[\![\boldsymbol{s}, t, \varphi \bowtie 0]\!] &:= \begin{cases} 1 \text{ if } [\![\boldsymbol{s}, t, \varphi]\!] \bowtie 0 \\ 0 \text{ otherwise} \end{cases} \\
[\![\boldsymbol{s}, t, \neg\chi]\!] &:= 1 - [\![\boldsymbol{s}, t, \psi]\!] \\
[\![\boldsymbol{s}, t, \chi_1 \wedge \chi_2]\!] &:= \min([\![\boldsymbol{s}, t, \chi_1]\!], [\![\boldsymbol{s}, t, \chi_2]\!])
\end{aligned}
$$

where $op : \mathbb{R}^n \to \mathbb{R}$ *are pointwise n-ary functions that map signals with finite variability property into signals with finite variability property,* $Agg_{t \in T}$ *is a placeholder for* $Int_{t \in T}$,$Min_{t \in T}$ *and* $Max_{t \in T}$ *which are the Riemann integral, minimum and maximum operator mapping real-valued functions into real values as* $Int_{t \in T}(s) = \int_T s(t)dt$*,* $Min_{t \in T}(s) = \min_{t \in T} s(t)$ *and* $Max_{t \in T}(s) = \max_{t \in T} s(t)$*, respectively.*

$$
F(t,T,\chi) = \begin{cases} [t,\tau] \text{ if exists the smallest } \tau \in t+T \wedge \chi(\tau) \\ [t,\tau+\varepsilon] \text{ with } \tau \in \inf\{t' \mid t' \in t+T \wedge \chi(t')\} \\ \emptyset \text{ otherwise} \end{cases}
$$

It is important to notice that $\bowtie 0$, and $\neg$ are two unary operators, respectively $\bowtie 0(x) := 1$ if $x \bowtie 0, 0$ otherwise and $\neg(x) := 1 - x$, as well as $\wedge$ is the minimum operator, i.e., $x \wedge y = \min(x, y)$. With a bit of redundancy, we decided to include them in the grammar because they allow us to define predicates.

A detailed description of the semantics follows.

Atomic variables. Each atomic variable X_i is interpreted over the corresponding i-th signal s_i of the trace $\boldsymbol{s}$ in a specific time t.

n-ary (pointwise) operators. The n-ary operators $op(\varphi_1, \ldots, \varphi_n)$ are interpreted with the corresponding op functions applied to the interpretation of

$\varphi_1, \ldots, \varphi_n$. We are considering only pointwise operators in $\mathbb{R}^n \to \mathbb{R}$ obtained by lifting operators defined in $\mathbb{R}$. As an example, consider the summation defined as $(f+g)(x) = f(x) + g(x)$.

Sliding Window Operator. The On-operator $\mathrm{On}_T(\mathrm{Agg}\,\varphi)$ is interpreted as the application of the aggregation function (Agg), that can be Min, Max or Int, on the signal generated by the interpretation of φ in $t+T$.

Find-first aggregation until. The find-first aggregation until $(\mathrm{Agg}\,\varphi)\mathrm{U}_T^d\chi$ is interpreted as the application of the Aggregation operator (Agg) on the signal generated by the interpretation of φ on a time window that is not fixed a priori but depends on the satisfaction of χ. This time window corresponding to $F(t,T,\chi)$ as defined above, starts in t and ends as soon as χ is satisfied in $t+T$. If all the time locations in $t+T$ do not satisfy χ, the operator returns the default value d.

Find-first pointwise until. The find-first pointwise until $\varphi\mathrm{U}_T^d\chi$ works as the previous one, but instead of performing a sliding window operator on the signal generated by the interpretation of φ, it returns the value of such interpretation on the upper bound of $F(t,T,\chi)$. If all the time locations in $t+T$ do not satisfy χ, and for this reason $F(t,T,\chi)$ is empty, the operator returns the default value d.

Negation and conjunction operators. Interpretation of negation and conjunction operators follows by considering them as unary and binary operators, respectively.

Filtered sliding window operator. The semantic definition of the filtered sliding window operator (Equation 1) poses a question. If indeed it is clear how `NaN` behaves with n-ary operators (see Section 2.2), we need to clarify how to interpret the aggregation on signals with undefined values. We opt for the following definition:

$$\mathrm{Agg}_T(\varphi \text{ if } \chi \text{ else } \texttt{NaN}) := \mathrm{Agg}_{T\cap\{t\in T|\chi(t)\}}\,\varphi$$

meaning that we are ignoring the values of signal φ where χ is false, which is coherent with the usual definition of filter. The well-foundedness of the previous definition depends on the aggregation function type and the structure of $T\cap\{t\in T \mid \chi(t)\}$. In Theorem 1 we show that the semantics of our logic preserves the finite variability property, meaning that χ has finite variability in any closed finite subset of $\mathbb{R}$, implying that $\{t\in T \mid \chi(t)\}$ consists in finite union of finite intervals, so that:

$$\mathrm{Agg}_{T\cap\{t\in T|\chi(t)\}}\,\varphi = \mathrm{Agg}_{\cup_{i\le k}T_i}\,\varphi = \langle Agg\rangle(\varphi|_{T_0},\ldots,\varphi|_{T_k})$$

with $\langle Min\rangle = \min$, $\langle Max\rangle = \max$, and $\langle Int\rangle = +$, which concludes the well-foundedness of the filtered sliding window operator definition.

3.1 Interpretation of atomic variables

In this paper, we interpret atomic variables either as piecewise-constant (pwc) functions, when modeling finite-state systems like gearshift dynamics that emit

events only on state changes, or as piecewise-linear continuous (pwlc) functions, when representing physical quantities measured over time. This distinction reflects common practice: pwc signals are used for discrete systems, while pwlc (or pwl) signals suit continuous measurements under sufficiently high sampling rates. Under this assumption, our semantics can produce only piecewise-constant functions or *piecewise-polynomial functions (pwp)*, provided we restrict ourselves to n-ary operators that map polynomials to polynomials.

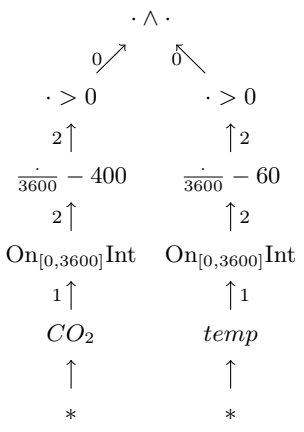
Since our semantics is defined recursively based on the semantics of simple formulae, it is crucial that the property of finite variability is preserved when applying logical operators. This preservation ensures the monitorability of the semantics and prevents the need to manage an infinite number of signal representations within a finite time horizon.

Theorem 1. *Let us consider a signal $\boldsymbol{s}$ with finite variability. For each formula $\varphi \in \mathcal{L}$, the semantics function $t \to [\![\boldsymbol{s}, t, \varphi]\!]$ is a signal with finite variability.*

The proof is by induction on the complexity of the formula and follows the demonstration already proposed for MITL [2], which is then adapted to STL. The difference here is the presence of the integral operator, which generates piecewise-polynomial functions instead of piecewise-linear or piecewise-constant functions. We provide the full proof in the Appendix A.1.

4 Monitoring

Our online monitoring framework is based on transducers that accept signals as inputs and generate output signals which may not be synchronized with the inputs, leading to possible delays. Following the approach in [11], we use the Lemire running maximum filter [17], which was adapted for piecewise-constant signals in [13]. We extend this approach to handle piecewise-continuous functions and present a modular transducer architecture named *Monitoring Graph (MG)*. Each node is a *Monitoring Unit (MU)* that receives signals as inputs, performs an internal computation, and generates signals as outputs. Based on the Direct Acyclic Graph structures, the generated signals are then forwarded to other MUs. As an example, please consider Figure 1, which reports the computational graph of property P1. There are two *leaf Monitoring Units* (ℓ-MUs), represented by the nodes CO_2 and *temp*, which receive as input measurement data and generate as output symbolic states with pwl functions. These symbolic states are then passed to MUs that perform integration ($\mathrm{On}_{[0,3600]}\mathrm{Int}$) without filtering and then to MUs that apply linear function operators ($\frac{\cdot}{3600} - 400$ and $\frac{\cdot}{3600} - 60$, respectively), and then to MUs that Booleanize the signal based on the inequality ($\cdot > 0$) and generate a pwc signal with the values 0 or 1 as output. These Boolean signals are then merged using the MUs ($\cdot \wedge \cdot$), which apply the min operator, and the resulting signal is provided as output. It is worth noting that, following our approach, online monitoring of specifications with future temporal operators can be performed without the use of *pastification* [26] and the need for past operators.



Algorithm 1 Step of monitoring unit $m = (Q, S, s_0, \lambda)$ receiving value v_{inp} on the i-th queue.

```
1: procedure m[i ← v_inp]
2:     ADD(Q[i], v_inp)
3:     while ISFULL(Q) do
4:         v_inp ← DEQUEUE(Q)
5:         s, V_out ← λ(s, v_inp)
6:         if V_out ≠ ∅ then
7:             for v_out ∈ V_out do
8:                 for j ∈ OE do
9:                     m[j ← v_out]
```

Figure 1: (Left) Monitoring graph of property $P1$: $\left(\frac{1}{3600}\mathrm{On}_{[0,3600]}\ \mathrm{Int}\ CO_2 > 400\right) \wedge \left(\frac{1}{3600}\mathrm{On}_{[0,3600]}\ \mathrm{Int}\ temp > 60\right)$. (Right) Computational step of a monitoring unit (Algorithm 1). The degree of monitoring units (Definition 9) is shown on output edges.

Definition 6 (Monitoring Units). *A Monitoring Unit (MU) $m = (Q, S, s_0, \lambda)$ is a tuple, where Q is an ordered list of n input (FIFO) queues that store symbolic states arriving from other MUs connected as input to m, S is the internal state (or memory) and s_0 is its initial value, $\lambda : S \times V^n \to S \times V$ is the state-output function that defines how the state and outputs are generated based on the current state, inputs and internal memory.*

In our framework, special monitoring units, called leaf-monitoring units (ℓ-MU), interpret the sampled input values as pwlc or pwc signals. In Figure 1, these units are represented by the nodes CO_2 and *temp*, which interpret inputs as pwlc signals.

Definition 7 (Leaf Monitoring Units). *A leaf monitoring unit (ℓ-MU) is a special MU with $|Q| = 1$, which accepts sample values $v = (t_v, f_v)$ where t_v is a time and f_v a value representing the measured or collected value at time t_v. It is a transducers that transform two consecutive sample values into symbolic states. In this paper, we define two types of ℓ-MU.*

- *ℓ^0-MU which return as output a pwc functions defined by (t_v, t_w, f_v)*
- *ℓ^1-MU which return as output a pwlc functions defined by $(t^v, t_w, \frac{f_w - f_v}{t_w - t_v}(t - t_v) + f_v)$*

where v and w represent two consecutive samples.

Definition 8 (Monitoring Graph). *A monitoring graph (MG) is a tuple (V, L, E, o) such that $(V \cup L, E)$ is a direct acyclic graph in which every vertex in V is a MU, and in L is a ℓ-MU, $o \in V$ is the terminal node.*

Below, we present how a monitoring graph is built from a formula and how the computations (i.e., what happens when a sample points arrive at a ℓ-MU) are performed.

Building Monitoring Graphs. The process begins by translating the *abstract syntax tree* of a formula into a monitoring tree by converting each logical operator into its corresponding monitoring unit. Identical sub-trees (if any) are merged to obtain the final monitoring graph. The leaf monitoring units are the input of this graph, while the node that was the root of the tree is the terminal node.

Computation of Monitoring Graphs. The computational steps of the entire graph are defined recursively, based on the computations of its individual MUs. When a new sample reaches an ℓ-MU, it is interpreted and forwarded to its connected MUs. These MUs receive the symbolic state, carry out their computations, and transmit their outputs to other connected MUs until a node without outgoing transitions is reached. Termination is guaranteed by the acyclic nature of the graph, and may or may not result in a final output. For simplicity, we do not include connection edges in the formal definition of an MU. Instead, we assume the existence of a routing mechanism that delivers outputs throughout the monitoring graph.

Computation of Monitoring Units (Algorithm 1). When a new value v_{inp} is received from the i-th input (line 1), it is added to the corresponding i-th input queue (line 2). If there is then at least one symbolic state in each input queue (line 3), the procedure DEQUEUE described below is executed in line 4. As a result, a vector $\boldsymbol{v}_{inp}$ of symbolic state with the same time intervals is generated and passed to the state-output function, which generates the new state (s) and a set of contiguous symbolic states (V_{out}) (line 5). If $V_{out} \neq \emptyset$, the monitoring unit propagates outputs through the rest of the monitoring graph (lines 7 – 9). This algorithm is generic and works for operators of any arity. However, it can be simplified for unary operators by keeping only lines 5 to 9 (and taking into account that the vector $\boldsymbol{v}_{inp}$ corresponds to the value v_{inp}).

Dequeue procedure (Algorithm 2). The dequeue procedure is a fundamental step in the calculation of a monitoring unit. It is responsible for retrieving values from the input queue (Q) and generating a vector input ($\boldsymbol{v}_{inp}$) that is then used by the output function (to generate a value that is passed to the rest of the monitoring graph). The Q is an ordered vector of queues, each storing a symbolic state as a top element with the same lower bound of the time interval as the others, but a different upper bound. For this reason, the first part of the procedure (lines 3 – 5) consists of identifying which is the minimum upper bound of those symbolic states ($\hat{t}$) so that it is always possible to extract a set of symbolic states with the same time bounds. This can be effectively done by removing the first chunk (of size depending on $\hat{t}$) of the symbolic state from each of the top symbolic states (lines 6 – 15). This new set of symbolic states corresponds to $\boldsymbol{v}_{inp}$, which is returned at line 16.

Algorithm 2 Dequeue procedure that considers as input the input queue (Q) of a monitoring unit.

procedure DEQUEUE(Q)
  $\boldsymbol{v}_{inp} \leftarrow []$, $\hat{t} \leftarrow -\infty$
  for $i < |Q|$ **do**
    $v \leftarrow$ PEAK($Q[i]$)
    $\hat{t} \leftarrow \min(\hat{t}, t_v^e)$
  for $i < |Q|$ **do**
    $v \leftarrow$ PEAK($Q[i]$)
    if $\hat{t} = t_v^e$ **then**
      $\boldsymbol{v}_{inp} \leftarrow \boldsymbol{v}_{inp} + [v]$
      POP($Q[i]$)
    else
      $v_{left} \leftarrow (t_v^s, \hat{t}, f_v)$
      $v_{right} \leftarrow (\hat{t}, t_v^e, f_v)$
      REPLACE($Q[i], v_{right}$)
      $\boldsymbol{v}_{inp} \leftarrow \boldsymbol{v}_{inp} + [v_{left}]$
  return $\boldsymbol{v}_{inp}$

Algorithm 3 Sliding window function customized for the Min aggregation function.

procedure SLIDE$_{MIN}$(T, v_l, s)
  $v_+, v_l \leftarrow$ SPLIT(T, v_l, s)
  $v'_+ \leftarrow$ TRANSFORM(v_+)
  $k \leftarrow n$
  while $s[k] \succ v'_+$ **do**
    REMOVE($s[k]$)
    $k \leftarrow k - 1$
  if $k > 0$ **then**
    $t' \leftarrow$ SOLVE($f_k(t) - f_+(a) = 0$)
    $s[k] \leftarrow ([a_k, t'), f_k)$
  else
    $t' \leftarrow a_0$
  $s[k+1] \leftarrow ([t', a_+), t \to f_+(a_+)))$
  if $\forall t \in [a_+, b_+]$, $\frac{\partial f_l}{\partial t} \leq 0$ **then**
    $s[k+1] \leftarrow ([t, b), t \to f(a_+))$
  else
    $s[k+2] \leftarrow ([a, b), f_+)$
  $v_{out} \leftarrow$ CUT(Δ_-, s)
  return v_{out}, v_l, s

Remark 2. It is important to note that we assume each first element of the input queues has the same time lower bound. It is simple to show that this property is an invariant for the dequeue procedure and that, at the beginning, we initialized each ℓ-MUs with a sample value $(-\infty, \texttt{NaN})$ so that all the queues are initialized with a symbolic state with time lower bound equal to $-\infty$. We are assuming that any function with input `NaN` will return `NaN`, as already discussed in Section 2.2.

We now describe all the monitoring units associated with the operators of our extended logic.

n-ary operator (op). This is a straightforward implementation of Algorithm 1 with empty memory and state-output function

$$\lambda((I, f_1), \ldots, (I, f_n)) := (I, op(f_1, \ldots, f_n))$$

All the other operators correspond to unary or binary application of Algorithm 1 where the λ function implements the sliding window algorithm or its customization. We now provide a general implementation of the sliding window algorithm, as well as custom implementations for the fixed-time window version (On-operator with Min/Max, Int) and the variable-time window version (for the two until operators).

Algorithm 4 Sliding window operator λ with window T and aggregator λ'.

1: **procedure** $\lambda(T, s, c, v)$
2: $V_{out} \leftarrow \emptyset$
3: $v_l \leftarrow v$
4: **while** $v_l \neq \emptyset$ **do**
5: $v_-, v_+, v_l, s \leftarrow \text{SLIDE}(T, v_l, s)$
6: $c, v_{\text{out}} \leftarrow \lambda'(c, v_-, v_+)$
7: $V_{\text{out}} \leftarrow V_{\text{out}} + [\text{SHIFT}(v_{\text{out}})]$
8: **return** V_{out}, s, c

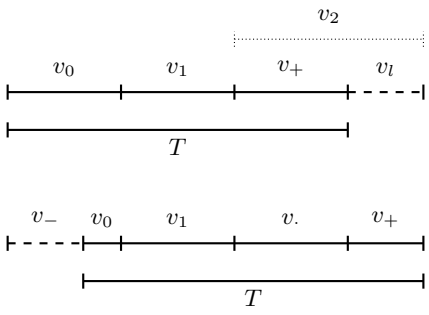


Figure 2: (Left) Sliding window algorithm. (Right) Visualization of the SLIDE procedure when v_l is added chunk by chunk in more than one loop.

Sliding Window Operator (Algorithm 4). The sliding window operator takes as input the window size T, the state s, which is the window queue storing symbolic states provided as inputs, the memory c, and the input symbolic state v. First, the set V_{out}, which will contain the resulting symbolic states, is initialized as the empty set (line 2), and v_l, which corresponds to the symbolic state we are trying to add to s, is initialized with v (line 3). While there is a symbolic state to add (line 4), the SLIDE procedure is executed (line 5). Based on the available space in the window queue s (which depends on the symbolic states already stored and the window size T), v_l can be added either completely or partially. In the first case, the SLIDE procedure returns $v_+ = v_l$ and $v_- = v_l = \emptyset$, meaning that the entire symbolic state is added to s. In the other case, v_l contains the portion that was rejected by s, v_+ is the portion of v_l that was added to s, and v_- is a portion of the first element in s that has been removed. Please refer to Figure 2 (Right) for a visual representation of this second case. At line 6, the aggregation function computes the resulting updated memory (c) and the symbolic state (v_{out}), based on the current memory and the added (v_+) and removed (v_-) symbolic states. At line 7, v_{out} is shifted backward by a time units (SHIFT procedure) and added to a vector storing all output symbolic states (V_{out}), and finally, at line 8, V_{out}, s, and c are returned.

Sliding Window Operator with Integral Aggregation ($On_T Int$). Monitoring of this sliding window operator is a straightforward adaptation of Algorithm 4. Given two symbolic states $v_- = ([a_-, b_-], f_-)$ and $v_+ = ([a_+, b_+], f_+)$, the update function $\lambda'(c, v_-, v_+)$ returns a symbolic state v_{out} and an update of the memory c corresponding to:

$$v_{out} = \left([a_-, b_-],\ t \to c - \int_{a_-}^{t} f_-(\tau)\, d\tau + \int_{a_+}^{t} f_+(\tau)\, d\tau \right) \tag{2}$$

$$c \leftarrow c - \int_{a_-}^{M} f_-(\tau)\, d\tau + \int_{a_+}^{M} f_+(\tau)\, d\tau \tag{3}$$

with $M := \min(b_- - a_-, b_+ - a_+)$.

Considering that the sliding window operator must behave as specified at the end of Section 3, a slight change to the Algorithm 4 is required to handle symbolic states with undefined values. When a symbolic state with an undefined value is received, it is treated as a symbolic state with a constant function equal to zero, and a counter is initialized to keep track of the duration of such symbolic states. This counter recognizes whether a sequence of symbolic states with undefined values spans a time window longer than the window length T. The operator must return a symbolic state with an undefined value in such cases.

Sliding Window Operator with Min/Max Aggregation ($On_T Min$/$On_T Max$). A naive implementation consists of memorizing all the symbolic states of the running window (in memory c) and evaluating min and max directly on it. Here we propose an adaptation of Lemire's running maximum filter algorithm [17] to our modeling of signals as symbolic states. The key idea is to work directly on the way symbolic states are stored in the state vector s of Algorithm 4 by maintaining the so-called *monotonic edge property*. In Algorithm 3, we propose a customized version of the SLIDE procedure (Algorithm 4, line 5) for the minimum aggregation function[4] which returns directly the result of the aggregation (v_{out}) and does not consider the λ' (Algorithm 4, line 6). The fundamental property of this algorithm is to maintain a state vector s (or window queue) such that

$$\min([v_0, \ldots, v_n]) = \min([v'_0, \ldots, v'_m]) = f'_0(a'_0) \tag{4}$$

where $v'_0 = ([a'_0, b'_0), f'_0)$ and $s = [v_0, \ldots, v_n]$ are the signal passed to the monitoring unit and considered for the aggregation function, and $[v'_0, \ldots, v'_m]$ is the derived signal from s which has the same aggregating result (first equality of Equation 4), but also has the monotonic edge property meaning that the result of the aggregation corresponds to its evaluation at the starting point (second equality of the same equation). The algorithm starts with the SPLIT procedure, which splits the input symbolic state (v_l) into the chunk added to the sliding window (v_+) and the chunk refused for the addition (stored using again v_l). At line 3, the TRANSFORM procedure will transform the $v_+ = ([a_+, b_+), f_+)$ into $v_+ = ([a_+, b_+), t \to f_+(b_+ - \epsilon))$ if $\forall t \in [a_+, b_+]$, $\frac{\partial f_l}{\partial t} \leq 0$, otherwise it will keep v'_+ as v_+. This transformation will make v'_+ itself satisfy the monotonic edge property, i.e., $\min_{t \in [a'_+, b'_+)} v'_+ = f'_+(a'_+)$. The algorithm continues by deleting all the symbolic states that are higher than the added v'_+. Since s satisfies the monotonic edge property, the while loop starts from the last element of s and goes backward until the first element not higher than v'_+ is identified (lines 5 – 7). If this element exists, the portion of the symbolic state $s[k]$ higher than

[4] The version for the maximum aggregation function is a straightforward adaptation of the version of the minimum aggregation version.

v'_+ is identified and removed (lines 8 – 10); otherwise, all the elements of the queue are removed. Finally, all the removed elements are replaced with a constant symbolic state from t' to the beginning of v'_+, corresponding to a_+ (line 13), and v'_+ is added as the last symbolic state of s. The output symbolic state of this procedure is identified by removing exactly the timed portion of the first symbolic state of s, which is $([a_0, a_0 + (b_+ - a_+), f_0])$ corresponding to the timed portion of the added symbolic state v'_+.

Remark 3. The optimized sliding window algorithm for computing the minimum and maximum assumes that the added symbolic state is either increasing, decreasing, or constant (both in the TRANSFORM procedure and at line 14). This is a property that must be enforced in order to use such an optimized procedure. Fortunately, this condition is naturally satisfied by leaf monitoring units of degree 0 and 1. It can also be preserved after applying an operator by evaluating the derivative of the signals and segmenting them based on whether this value is greater than or less than zero.

Find-first aggregation until $\psi U^d_T \chi$ and Find-first pointwise until $\varphi \downarrow U^d_T \chi$. This is a simple variation of the sliding window operator where the time window length is fixed. We report its description in the Appendix A.2.

The monitoring procedure described above sometimes requires finding the roots of polynomials. This occurs both in the evaluation of unary inequalities and when computing the minimum or maximum, both as binary operators or as an aggregation function of a sliding window operator. Since root-finding for polynomials of degree higher than 4 can be computationally expensive, often requiring approximation methods, and such approximations may compromise monitoring precision, we choose to consider only monitoring graphs where root-finding can be achieved algebraically, in closed form. For this purpose, it is convenient to introduce the following definition.

Definition 9 (Edge degree). *We introduce the degree function $deg : \mathcal{V}_{M_G} \to \mathbb{N}$ that associates to each edge of a monitoring graph (M_G) a natural number. It is defined recursively as follows:*

$$deg((m_a, m_b)) = p \text{ if } m_a \in \ell^p\text{-}MU$$
$$deg((m_a, m_b)) = \beta(m_a)(deg((m_0, m_a)), \ldots, deg((m_n, m_a)))$$

where $\{(m_i, m_a)\}_{i \leq n}$ are the incoming edges of the monitoring unit m_a, ℓ^p-MU is the set of leaf monitoring units which interpret time-stamped values as pwp of degree p, and $\beta : \mathcal{M}_G \to (\mathbb{N}^* \to \mathbb{N})$ is a function taking as input a monitoring unit m (associated to an operator of arity, e.g. k) and producing as output a function $\beta(m) : \mathbb{N}^k \to \mathbb{N}$ which computes the degree of the polynomial generated as output, provided k polynomials as inputs.

Please consider Figure 1 (Right) as example, where the degree of edges has been added. For more examples, refer to Table 1 in Appendix A.3, where some beta functions have been reported.

Computational Complexity of the monitoring algorithm. Our algorithm's time complexity grows linearly with the number of symbolic states processed and with the size of the Monitoring Graph. Indeed, all monitoring units use the received symbolic states only once. This is also valid for the sliding-window operators for computing minima and maxima, as they rely on Lemire's optimal algorithm. Finally, the sliding-window integration operator is also linear with the number of received symbolic states, as we limit our attention to cases where the integration is purely symbolic.

5 Implementation and Experiments

We demonstrate our monitoring algorithm, implemented in Python and available online [23], on the two running examples.

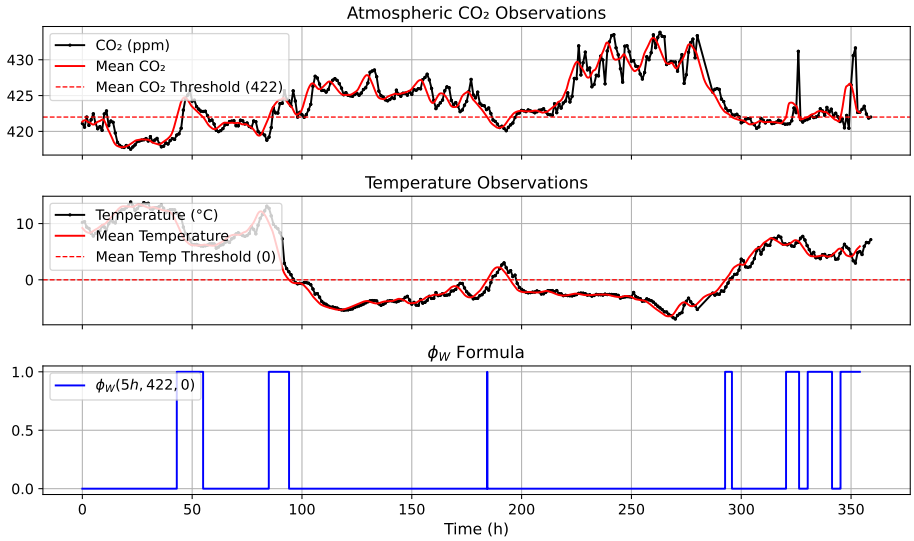


Figure 3: Weather Case Study - The top chart displays the level of CO_2 in ppm, its rolling mean corresponding to evaluation of $\frac{1}{T}\text{On}_{[0,T]}$ Int CO_2, and the relative threshold of 422 ppm. The middle chart reports the level of air temperature in Celsius degrees, its rolling mean corresponding to evaluation of $\frac{1}{T}\text{On}_{[0,T]}$ Int *temp*, and the relative threshold of 0 °C. Finally, the bottom chart displays the evaluation of property $\varphi_W(5h, 422, 0)$.

Weather Events. This use case corresponds to the first running example. In weather monitoring, a typical requirement is defined by the property $\varphi_W(T, \theta_1, \theta_2)$ (P1, Section 1), which is typically used to specify a possible fire event in an outdoor environment, commonly with $\theta_1 = 400$ ppm and $\theta_2 = 60\,^{\circ}\text{C}$, or an indoor environment to define an air-quality constraint typically with $\theta_1 = 1000$ ppm and

$\theta_2 = 25\,^{\circ}\mathrm{C}$, each evaluated over an appropriate time window. In this case, we use real monitoring data collected at the Hohenpeissenberg, Germany, weather station, which is part of the Integrated Carbon Observatory System (ICOS) network [16]. In Figure 3 we report results for monitoring of the property $\varphi_W(5h, 422, 0)$ and provide a video of the monitoring algorithm acting at runtime in the GitHub repository [23].

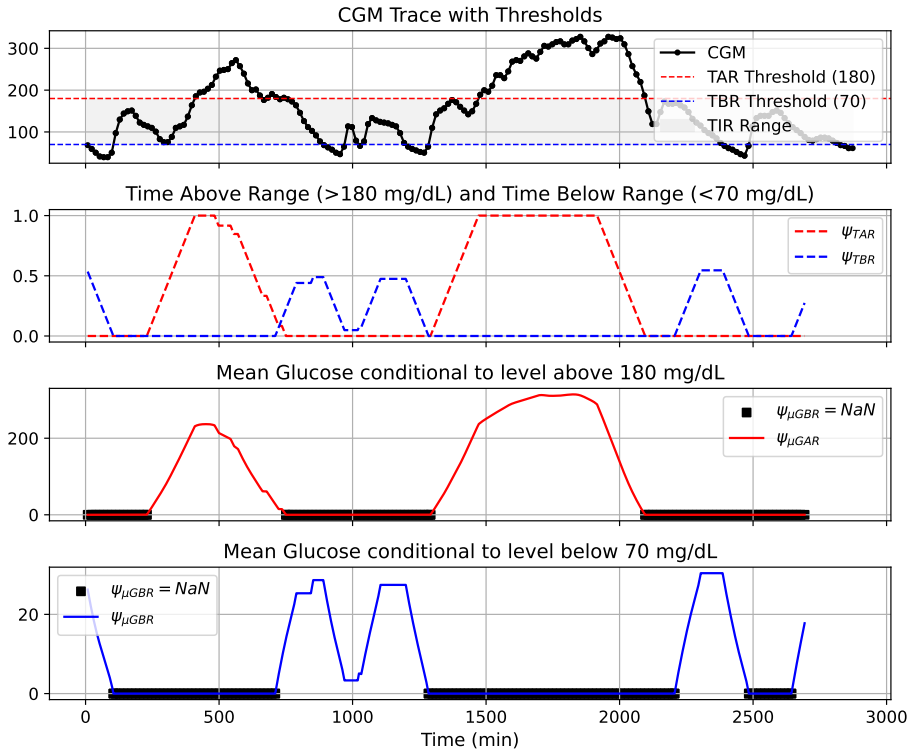


Figure 4: CGM case study - The first chart shows glucose levels with Time in Range (TIR), Time Above Range (TAR), and Time Below Range (TBR) thresholds. The second shows evaluations of ψ_{TAR} and ψ_{TBR}. The third and fourth show $\psi_{\mu GAR}$ and $\psi_{\mu GBR}$, along with the portions of the time domain (represented with black squares) where these evaluations cannot be determined because the filter conditions are not satisfied across the entire sliding window.

Artificial Pancreas. This follows the second running example. We simulate the monitoring and evaluation of a deployed AP controller using the ShanghaiT1DM dataset [27], a publicly available collection of real-world clinical and monitoring data of adult patients with type 1 diabetes mellitus (T1DM), in Shanghai, China. We analyze the temporal evolution of glucose levels for a single patient over two

consecutive days. Measurements are recorded every 15 minutes and modeled as a piecewise linear signal. Figure 4 presents the glucose trace alongside safe thresholds defining the Time in Range (TIR). The derived metrics ψ_{TAR} and ψ_{TBR}, representing the percentage of time spent above and below the safe range, respectively, and the mean glucose level conditioned during the occurrence of such events associated with Hyperglycemia and Hypoglycemia $\psi_{\mu GAR}$ and $\psi_{\mu GBR}$, are shown. These analyses offer valuable insight into both the severity and progression of glycemic events. For instance, we can observe that the second episode of hyperglycemia is not only longer but also more severe in terms of average glucose levels. For a demonstration video of the monitoring algorithm at runtime, refer to the GitHub repository [23].

6 Conclusions

We extend the specification language of Bakhirkin and Basset [3] by adding the integral operator as an aggregation function, allowing for the definition of cumulative properties and introducing also a filtering operator to constrain aggregation where specified conditions hold. We also generalize from piecewise-constant (pwc) to piecewise-polynomial (pwp) signals. Furthermore, we leverage transducers to propose a modular architecture for online monitoring of the extended logic, allowing property evaluation during target system execution. The proposed implementation of this architecture has a minor limitation: it may introduce delays in producing verdicts depending on the formula's time horizon, along with a trade-off in computational performance.

There are several directions for future work. First, we resolve the monitoring of future operators with the introduction of delays, meaning that we need to wait the entire time horizon of the formula to produce a verdict. It would be interesting to implement an interval value semantics that bounds the evaluation of the future operator at least for the maximum and minimum aggregation functions. Second, we currently restrict our monitoring framework to Monitoring Graphs so that the evaluation of the semantics is symbolic, meaning that no approximation technique is needed to evaluate zeros of polynomials (for example, when their degree is higher than 4). A promising direction is to incorporate efficient approximation routines when symbolic evaluation is infeasible. Third, a tool paper version of this contribution could be considered by including a more in-depth case study and a thorough comparison with stream-based approaches such as RTLola [14] and TeSSLa [18], as well as tools like VeriMon [5] and MonPoly [6] for monitoring discrete-time logic properties.

Acknowledgment. This study was carried out within the PRIN project n. 20228FT78M DREAM (modular software design to reduce uncertainty in ethics-based cyber-physical systems).

References

1. Akazaki, T., Hasuo, I.: Time robustness in MTL and expressivity in hybrid system falsification. In: Proc. of CAV. pp. 356–374. Springer (2015)
2. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. Journal of the ACM (JACM) **43**(1), 116–146 (1996)
3. Bakhirkin, A., Basset, N.: Specification and efficient monitoring beyond STL. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 79–97. Springer (2019)
4. Bakhirkin, A., Ferrère, T., Henzinger, T., Nickovic, D.: The first-order logic of signals. In: International Conference on Embedded Software (EMSOFT) (2018)
5. Basin, D., Dardinier, T., Hauser, N., Heimes, L., Huerta y Munive, J.J., Kaletsch, N., Krstić, S., Marsicano, E., Raszyk, M., Schneider, J., et al.: VeriMon: a formally verified monitoring tool. In: International Colloquium on Theoretical Aspects of Computing. pp. 1–6. Springer (2022)
6. Basin, D.A., Klaedtke, F., Zalinescu, E.: The MonPoly monitoring tool. RV-CuBES **3**, 19–28 (2017)
7. Battelino, T., et al.: Clinical targets for continuous glucose monitoring data interpretation: Recommendations from the international consensus on time in range. Diabetes Care **42**(8), 1593–1603 (2019)
8. Brim, L., Dluhoš, P., Šafránek, D., Vejpustek, T.: STL∗: Extending signal temporal logic with signal-value freezing operator. Information and computation **236**, 52–67 (2014)
9. Buyukkocak, A.T., Aksaray, D., Yazıcıoğlu, Y.: Control synthesis using signal temporal logic specifications with integral and derivative predicates. In: 2021 American Control Conference (ACC). pp. 4873–4878. IEEE (2021)
10. Clifford: Preliminary sketch of biquaternions. Proceedings of the London Mathematical Society **1**(1), 381–395 (1871)
11. Deshmukh, J.V., Donzé, A., Ghosh, S., Jin, X., Juniwal, G., Seshia, S.A.: Robust online monitoring of signal temporal logic. Formal Methods in System Design **51**, 5–30 (2017)
12. Divani, M., et al.: Assessment of hyperglycemia, hypoglycemia and inter-day glucose variability using continuous glucose monitoring among diabetic patients on chronic hemodialysis. Journal of Clinical Medicine **10**(18), 4116 (2021)
13. Donzé, A., Ferrere, T., Maler, O.: Efficient robust monitoring for STL. In: International conference on computer aided verification. pp. 264–279. Springer (2013)
14. Faymonville, P., Finkbeiner, B., Schledjewski, M., Schwenger, M., Stenger, M., Tentrup, L., Torfah, H.: StreamLAB: stream-based monitoring of cyber-physical systems. In: International Conference on Computer Aided Verification. pp. 421–431. Springer (2019)
15. Hoogendoorn, C.J., et al.: Dynamic relationships among continuous glucose metrics and momentary cognitive performance in diverse adults with type 1 diabetes. Diabetes Care **48**(5), 799–806 (2025)
16. Kubistin, D., Plaß-Dülmer, C., Arnold, S., Kneuer, T., Lindauer, M., Müller-Williams, J., Schumacher, M., ICOS RI: Icos atmosphere level 2 data, hohenpeissenberg, release 2024-1 (2024). `https://doi.org/10.18160/QSDJ-P768`, `https://meta.icos-cp.eu/collections/jXIGuaTJJapMNE8XKAP4tC_d`
17. Lemire, D.: Streaming maximum-minimum filter using no more than three comparisons per element. arXiv preprint cs/0610046 (2006)

18. Leucker, M., Sánchez, C., Scheffel, T., Schmitz, M., Schramm, A.: TeSSLa: runtime verification of non-synchronized real-time streams. In: Proceedings of the 33rd Annual ACM Symposium on Applied Computing. pp. 1925–1933 (2018)
19. Ma, M., Preum, S.M., Ahmed, M.Y., Tärneberg, W., Hendawi, A., Stankovic, J.A.: Data sets, modeling, and decision making in smart cities: A survey **4**(2) (Nov 2019). https://doi.org/10.1145/3355283
20. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems. pp. 152–166. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
21. Mamouras, K., Chattopadhyay, A., Wang, Z.: A compositional framework for algebraic quantitative online monitoring over continuous-time signals. International Journal on Software Tools for Technology Transfer **25**(4), 557–573 (2023)
22. Pnueli, A., Zaks, A.: On the merits of temporal testers. 25 Years of Model Checking: History, Achievements, Perspectives pp. 172–195 (2008)
23. Silvetti, S., Loreti, M., Nenzi, L.: Github repository (2025), https://github.com/LogArtLab/gemon
24. Silvetti, S., Nenzi, L., Bartocci, E., Bortolussi, L.: Signal convolution logic. In: International Symposium on Automated Technology for Verification and Analysis. pp. 267–283. Springer (2018)
25. Tsigkanos, C., Bersani, M.M., Frangoudis, P.A., Dustdar, S.: Edge-based runtime verification for the internet of things. IEEE Transactions on Services Computing **15**(5), 2713–2727 (2022). https://doi.org/10.1109/TSC.2021.3074956
26. Yamaguchi, T., Hoxha, B., Ničković, D.: Rtamt–runtime robustness monitors with application to cps and robotics. International Journal on Software Tools for Technology Transfer **26**(1), 79–99 (2024)
27. Zhao, Q., Zhu, J., Shen, X., Lin, C., Zhang, Y., Liang, Y., Cao, B., Li, J., Liu, X., Rao, W., et al.: Chinese diabetes datasets for data-driven machine learning. Scientific Data **10**(1), 35 (2023)

A Appendix

A.1 Proof of Theorem 1

Proof. We proceed by induction on the complexity of the formula. For simplicity, let us denote the semantics function with $[\![\boldsymbol{s}, \cdot, \varphi]\!]$.

Atomic Variables. It is obvious because for the hypothesis we interpret them as (pwc or pwlc) signal of finite variability.

n-ary operator (op). For the hypothesis, we restrict to operators that map signals with finite variability to a signal with finite variability. Since we consider specific operators such as min, max, the inequality operator, i.e., $\bowtie 0$, and more generally algebraic operators (i.e., see definition of the negation), we provide for each of them a proof.

1. Algebraic operators – We provide a proof for binary operators. Extension to unary or n-ary operator with $n \geq 3$ is straightforward. Let $op(u, v)|_T$ be the restriction of $op(u, v)$ to an arbitrary subset of $\mathbb{R}_{\geq 0}$ obtained by applying op to signal u and v. Since op is a pointwise operator, we have that $op(u, v)|_T = op(u|_T, v|_T)$. For inductive hypotheses $u|_T$ and $v|_T$ admit both a representation as a finite succession of symbolic states. As reported in Remark 1, it is always possible to define two equivalent representations $u'|_T$ and $w'|_T$ with corresponding time intervals. So we have that $op(u, v)|_T = op(u'|_T, w'|_T) = \{(I_i, op(f_{u_i}, f_{v_i}))\}$ which is a finite succession of symbolic states. Considering that for this case, the operators are algebraic, it is clear that $(I_i, op(f_{u_i}, f_{v_i}))$ is resolved because the resulting function cannot be divided into two different functions.
2. Inequality operator ($\bowtie 0$) – Let us consider for inductive hypothesis that the signal $v = (v_1, \ldots, v_n)$ satisfy the finite variability property meaning that $n < +\infty$. For definition, we have that $v \bowtie 0 = (v_1 \bowtie 0, \ldots, v_n \bowtie 0)$. Now v_i are symbolic states generated by applying a maximum number of operators depending on φ, let us say k. Since we interpret variables as pwc or pwl signals, and each application of those operators can increase the degree of the function until a maximum depending on k, each v_i can be refined with a maximum number of adjacent symbolic states (with constant Boolean function) depending on the number of zeros of such polynomials. It proves that $v \bowtie 0$ satisfies the finite variability property.
3. Min and Max binary operator – We prove for the minimum operator. Demonstration follows considering its definition i.e., $\min(x, y) = x$ if $x - y > 0$ else y and that, following the demonstration for algebraic operators, we obtain that $min(u, v) = (I_i, \min(f_{u_i}, f_{v_i}))$. Now we consider that for each symbolic states $\min(f_{u_i}, f_{v_i})$ satisfies the finite variability property because, for definition, it depends only on the finite variability property of $f_{u_i} - f_{v_i}$ which holds since it is an algebraic expression.

Temporal operators. We provide a demonstration for a generic temporal operator. First, we consider that a temporal operator can be generalized as an aggregation function acting on a specific fixed time window for the On_T operator or variable (but bounded) time windows for the until operators. Let us consider the case of $\mathrm{On}_{[a,b]}(v)$ for the definition if we restrict it to a finite subset of $\mathbb{R}_{\geq 0}$ let's say $[c, d]$ then this signal is the result of applying an aggregating function to v in the set $[c-a, d-b]$. In a more general description, any finite portion of a signal $\mathrm{On}_{[a,b]}(v)$ is generated by a finite portion of v. For the inductive hypothesis, v restricted to $[c-a, d-b]$ satisfies the finite variability property meaning that can be written as $(v_1, v_2, \ldots, v_k)$. Now, for any time window, the temporal operator will admit a finite number of possible slides (Algorithm 4, line 5). Each slide procedure generates two symbolic states v_- and v_+ (Algorithm 4, line 6), which are then passed to a state-output function λ' that generates the resulting symbolic state. We need to show that λ' generates symbolic states that admit a finite representation as adjacent resolved symbolic states. This is the case both for the integral operator, where λ' is reported in Equation 2, and for min and max sliding operators, where λ' consists of applying those operators between a finite number of symbolic states.

A.2 Monitoring

Find-first aggregation until $\psi\, U^d_T \chi$. The find-first aggregation-until operator generalizes fixed-length sliding windows by permitting windows of variable duration. Please consider that if the time bound of the operator is $T = [a, b]$, we need to keep a buffer time window (s) of length b. Let's identify with W_a the actual starting time of such buffer time window. Whenever a symbolic state associated with ψ (denoted s_ψ) or with χ (denoted s_χ) arrives at the monitoring unit, if s_χ does not satisfy χ then s_ψ is accumulated in the buffer time window s. Two outcomes are then possible: if the buffer reaches its time bound while χ remains false, the monitor emits an output v_{out} whose time interval matches that of the oldest buffered state v_- but whose value is the constant default d; alternatively, if $s_\chi = ([t_a, t_b], f)$ arrives and $t_a \in [W_a + a, W_a + b]$ the buffer time windows is flushed of $t_a - (W_a + a)$ time units by removing each stored state in order, applying the aggregation function to it, and emitting the resulting v_{out}. In this way, the operator either returns the default d when the window closes without any χ, or yields the computed aggregation immediately upon the first occurrence of χ.

Find-first pointwise until $\varphi \downarrow U^d_T \chi$. This operator mirrors the find-first aggregation-until; the difference is that no aggregation is performed. Indeed, when the portion of the time window buffer is flushed, no aggregation is computed. Still, the symbolic states are returned directly after having substituted their function value with the constant function equal to $f_{s_\psi}(a)$.

A.3 Beta Functions

In the following Table, we report the β function of common operators.

Operator description	**Operator formula**	$\boldsymbol{\beta(n_0, \dots)}$
Multiplication by constant	$c \cdot p$	n
Addition by constant	$c + p$	n
Power elevation	p^m	n^m
Inequality	$p \bowtie 0$	0
Multiplication	$p_1 \cdot p_2$	$\max(n_1, n_2)$
Addition	$p_1 + p_2$	
Minimum	$\min(p_1, p_2)$	
Maximum	$\max(p_1, p_2)$	
Timewindow integral	$\mathrm{On}_T\mathrm{Int}$	$n + 1$
Timewindow min	$\mathrm{On}_T\mathrm{Min}$	n
Timewindow max	$\mathrm{On}_T\mathrm{Max}$	n

Table 1: The table presents β-functions for selected monitoring units, categorized by their associated operators. For instance, the unit m that computes the sum of two input signals is characterized by $\beta(m)(n_1, n_2) = \max(n_1, n_2)$, where n_1 and n_2 are the degrees of the input polynomials.