Specification and Efficient Monitoring Beyond STL

Alexey Bakhirkin, Nicolas Basset

Specification and Efficient Monitoring Beyond STL

Alexey Bakhirkin and Nicolas Basset

Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, 38000 Grenoble, France

Abstract. An appealing feature of Signal Temporal Logic (STL) is the existence of efficient monitoring algorithms both for Boolean and real-valued robustness semantics, which are based on computing an aggregate function (conjunction, disjunction, min, or max) over a sliding window. On the other hand, there are properties that can be monitored with the same algorithms, but that cannot be directly expressed in STL due to syntactic restrictions. In this paper, we define a new specification language that extends STL with the ability to produce and manipulate real-valued output signals and with a new form of until operator. The new language still admits efficient offline monitoring, but also allows to express some properties that in the past motivated researchers to extend STL with existential quantification, freeze quantification, and other features that increase the complexity of monitoring.

1 Introduction

Signal Temporal Logic (STL [16,17]) is a temporal logic designed to specify properties of real-valued dense-time signals. It gained popularity due to the rigour and the ability to reason about analog and mixed signals; and it found use in such domains as analog circuits, systems biology, cyber-physical control systems (see [3] for a survey). A major use of STL is in monitoring: given a signal and an STL formula, an automated procedure can decide whether the formula holds at a given time point.

Monitoring of STL is reliably efficient. A monitoring procedure typically traverses the formula bottom up, and for every sub-formula computes a satisfaction signal, based on satisfaction signals of its operands. Boolean monitoring is based on the computation of conjunctions and disjunctions over a sliding window ("until" is implemented using a specialized version of running conjunction), and robustness monitoring (computing how well a signal satisfies a formula [10,9]) is based on the computation of minimum and maximum over a sliding window. The complexity of both Boolean and robustness monitoring is linear in the length of the signal and does not depend on the width of temporal windows appearing in the formula. At the same time, for a range of applications, pure STL is either not expressive enough or difficult to use, and specifying a desired property often becomes a puzzle of its own. The existence of robustness and other real-valued semantics does not always help, since a monitor can perform a limited set of operations that the semantics assigns to Boolean operators. For example, for robustness semantics, min and max are the only operations beyond the atomic proposition level.

This work was partially supported by the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement nr. 306595 "STATOR".

One way to work around the expressiveness issues of STL is pre-processing: a computation that cannot be performed by an STL monitor can be performed by a pre-processor and supplied as an extra input signal. For a number of reasons, this is not always satisfactory. First, for monitoring of continuous-time signals, there is a big gap between the logical definitions of properties and the implementation of monitors. In continuous-time setting, properties are defined using quantification, upper and lower bounds, and similar mathematical tools for dense sets, while a monitor works with a finite piecewise representation of a signal and performs a computation that is based on induction and other tools for discrete sets. Leaving this gap exposed to the user, who has to implement the pre-processing step, is not very user-friendly. Second, monitoring of some properties cannot be cleanly decomposed into a pre-processing step followed by standard STL monitoring. Later, we give a concrete example using an extended "until" operator, and for now, notice that "until" instructs the monitor to compute a conjunction over the window that is not fixed in advance, but is defined by its second operand. Because of that, multiple researches have been motivated to search for a more expressive superset of STL that would allow to specify the properties they were interested in.

One direction for extension is to add to the original quantifier-free logic (MTL, STL) a form of variable binding: a freeze quantifier as in STL* [6], a clock reset as in TPTL [1], or even first order quantification [2]. Unfortunately, such extensions are detrimental to complexity of monitoring. When monitoring logics with quantifiers using standard bottom-up approach, subformulas containing free variables evaluate not to Boolean- or real-valued signals, but to maps from time to non-convex sets, and they cannot in general be efficiently manipulated (although for some classes of formulas monitoring of logics with quantifiers works well [4,13]). Perhaps the most benign in this respect but also least expressive extension is 1-TPTL (TPTL with one active clock), which is as expressive as MITL, but is easier to use and admits a reasonably efficient monitoring procedure [11].

An alternative direction is to define a quantifier-free specification language with more flexible syntax and sliding window operations. For example, Signal Convolution Logic (SCL [20]) allows to specify properties using convolution with a set of select kernels. In particular, it can express properties of the form "statement φ holds on an interval for at least X% of the time". In SCL, every formula has a Boolean satisfaction signal, but some works go further and allow a formula to produce a real-valued output signal based on the real-valued signals of its subformulas. This already happens for robustness of STL in a very limited way, and can be extended. For example, [19] presents temporal logic monitoring as filtering, which allows to derive multiple different real-valued semantics. Another work [7] focuses on the practical application of robustness in falsification and allows to choose between different possible robust semantics for "eventually" and "always", in particular to replace min or max with integration where necessary.

This paper is our take on extending STL in the latter direction. We define a specification language that is more expressive than STL, but not less efficient to monitor offline, i.e., the complexity of monitoring is linear in the length of the signal and does not depend on the width of temporal windows in the formula (the latter property tends to be missing from the STL extensions, even when the authors can achieve linear complexity for a fixed formula). The most important features of the new language are as follows.

1. We remove several syntactic constrains from STL: we allow a formula to have a real-valued output signal; we allow these signals to be combined in a point-wise way with arithmetic operations, comparisons, etc. This distinguishes us from the works that use standard MTL or STL syntax and assign them new semantics [10,19].
2. We allow to apply an efficiently computable aggregate function over a sliding window. We currently focus on min and max, which are enough to specify properties that motivated the development of more expressive and hard to monitor logics.
3. We offer a version of "until" operator that performs aggregation over a sliding window of dynamic width, that depends on satisfaction of some formula. This distinguishes us from the works that focus on aggregation over a fixed window [20].

Finally, we focus our attention on continuous-time piecewise-constant and piecewise linear signals; we describe the algorithms and prepare an implementation only for piecewise-constant.

2 Motivating Examples

Before formally defining the new 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 (Stabilization) The first interesting property is stabilization 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 formalize 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 [2]), 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 $\mathrm{On}_{[a,b]}$ to define the temporal window of constant width and the operators Min and Max (capitalized) to denote the minimum and maximum over the previously defined window.
Signal x stabilizes within 0.05 *units of an unknown value for* 200 *time units:*

$$\mathrm{On}_{[0,200]}\,\mathrm{Max}\,x - \mathrm{On}_{[0,200]}\,\mathrm{Min}\,x \leq 0.1$$

Fig. 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 siding 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.

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 [6], 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.

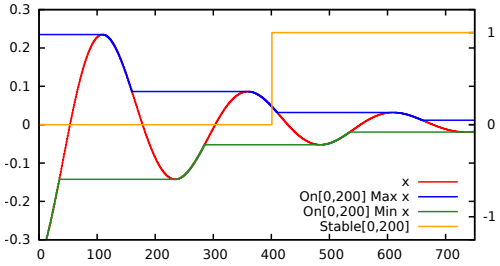

Fig. 1. Damped oscillation $x(t)$ and its maximum and minimum over the window $[t, t + 200]$.

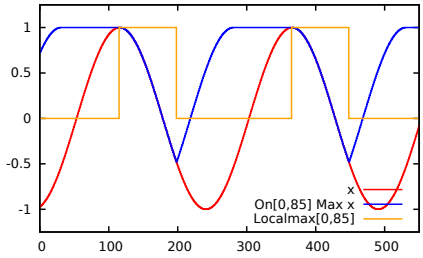

Fig. 2. Sine wave $x(t)$, its maximum over the window $[t, t + 200]$, and whether $x(t)$ is a local maximum on the interval $[t, t + 200]$.

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

$$x \geq \mathrm{On}_{[0,85]}\ \mathrm{Max}\ x$$

Fig. 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 siding 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.

Example 3 (Stabilization 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:*

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

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 $2.5 \cdot \mathrm{T}$ stays within some bounds.
Signal x increases approximately with slope 2.5 *during the next* 100 *time units:*

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

3 Syntax and Semantics

From the examples above we can foresee how the new language looks like. Formally, an *(input) signal* is a function $w : \mathbb{T} \to \mathbb{R}^n$, where the time domain $\mathbb{T}$ is a closed real

interval $[0, |w|] \subseteq \mathbb{R}$, and the number $|w|$ is the *duration* of the signal. We refer to signal components using their own letters: $x, y, \cdots \in \mathbb{T} \to \mathbb{R}$. We assume that every signal component is piecewise-constant or piecewise-linear.

The semantics of a formula is a piecewise-constant or piecewise-linear function from real time (thus, has real-valued switching points) to a dual number (rather than a real). We defer the discussion of dual numbers until Section 3.2; for now we note that they extend reals, and a dual number can be written in the form $a + b\varepsilon$, which, when $b \neq 0$, denotes a point infinitely close to a. We denote the set of dual numbers as $\mathbb{R}_\varepsilon$. Our primary use of a dual number is to represent a time point strictly after an event (switching point, threshold crossing, etc) but before any other event can happen; as a result we have to allow an output signal to have a dual value, denoting a value that is attained at this dual time point.

Syntax We can write the abstract syntax of our language as follows:

$$\begin{aligned} \varphi &::= c \mid x \mid f(\varphi_1 \cdots \varphi_n) \mid \mathrm{On}_{[a,b]}\, \psi \mid \psi\ \mathrm{U}^d_{[a,b]}\ \varphi \mid \varphi_1 \downarrow \mathrm{U}^d_{[a,b]} \varphi_2 \\ \psi &::= \mathrm{Min}\, \varphi \mid \mathrm{Max}\, \varphi \end{aligned} \tag{1}$$

where c is a real-valued constant; x refers to an input signal; f is a real-valued function symbol (e.g., sum, absolute value, etc); for the On-operator, a and b can be real numbers or (with some abuse of notation) $\pm\infty$, i.e., the interval may refer to both past and future, bounded or unbounded; for the U-operator, d is a real value, and a, b are non-negative, and b can be ∞, i.e., the interval refers to bounded or unbounded future. Let us go over some of the features of the new language and then formally write down its semantics.

Point-wise Functions Function symbol f ranges over real-valued functions $\mathbb{R}^n \to \mathbb{R}$ that preserve the chosen shape of signals (and can be lifted to dual numbers). In this paper, we focus on piecewise-constant and piecewise-linear signals, so when f is applied point-wise to a piecewise-constant input, we want the result to be piecewise-constant; when f is applied point-wise to a piecewise-linear input, we want the result to be piecewise-linear. Examples of such functions are addition, subtraction, min and max of finitely many operands (we use lowercase min and max to denote a real-valued n-ary function), multiplication by a constant, absolute value, etc.

Boolean Output Signals Output signals of some formulas can informally be interpreted as Boolean-valued. In Example 2, "x" and "$\mathrm{On}_{[0,85]}\, \mathrm{Max}\, x$" are dual-valued, but the result of their comparison, "$x \geq \mathrm{On}_{[0,85]}\, \mathrm{Max}\, x$" should be interpreted as Boolean. Here, we take the more simple path and treat a Boolean signal as a special case of a real-valued signal that can take the value of 0 or 1. We expect comparison operators to produce a value in $\{0, 1\}$, e.g., $\varphi_1 \leq \varphi_2$ is a shortcut for "if $\varphi_1 \leq \varphi_2$ then 1 else 0". Standard Boolean connectives can then be defined as follows:

$$\varphi_1 \wedge \varphi_2 = \min\{\varphi_1, \varphi_2\} \qquad \varphi_1 \vee \varphi_2 = \max\{\varphi_1, \varphi_2\} \qquad \neg\varphi = 1 - \varphi$$

Another option would be to distinguish Boolean-valued formulas on the syntactic level.

Temporal φ-Formulas Symbol φ denotes a temporal formula that has a dual-valued output signal. In other words, it can be evaluated at a time point and produces a dual value. A φ-formula may:

1. refer to an input signal x;
2. apply a real-valued function f pointwise to the outputs its φ-subformulas;
3. apply an aggregate function over the sliding window $[a, b]$ (with some abuse of notation a can be be $-\infty$, and b can be ∞);
4. be an "until" formula, which is described in Section 3.3.

Interval ψ-Formulas 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 [15,9], and preserve the piecewise-constant and piecewise-linear shapes. In discrete time or for piecewise-polynomial signals, we could use more aggregate operations, e.g., integration.

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

$$\mathrm{F}_{[a,b]}\,\varphi = \mathrm{On}_{[a,b]}\,\mathrm{Max}\,\varphi \qquad\qquad \mathrm{G}_{[a,b]}\,\varphi = \mathrm{On}_{[a,b]}\,\mathrm{Min}\,\varphi$$

3.1 Semantics of Until-Free Fragment

The semantics of the until-free fragment is straightforward. The semantics of a φ-formula is a function $[\![\varphi]\!] : \mathbb{T} \to \mathbb{R}_\varepsilon$ mapping real time to a dual value. We define it as:

$$\begin{aligned} &[\![x]\!](t) = x(t) \qquad\qquad [\![\mathrm{On}_{[a,b]}\,\psi]\!](t) = [\![\psi]\!]([t+a, t+b]) \\ &[\![f(\varphi_1 \ldots \varphi_n)]\!](t) = f([\![\varphi_1]\!](t) \ldots [\![\varphi_n]\!](t) \end{aligned} \tag{2}$$

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 semantics of a ψ-formula is a function $[\![\psi]\!] : (\mathbb{R} \cup -\infty) \times (\mathbb{R}_\varepsilon \cup \infty) \to \mathbb{R}_\varepsilon$ from an interval of time with real lower bound to a dual value. The upper bound of the interval can be dual-valued, which will be used by the "until" operation (see Section 3.3).

$$[\![\mathrm{Min}\,\varphi]\!][a, b] = \min_{[a,b]} [\![\varphi]\!] \qquad [\![\mathrm{Max}\,\varphi]\!][a, b] = \max_{[a,b]} [\![\varphi]\!] \tag{3}$$

The way we define min and max over an interval for a discontinuous piecewise-linear function relies on dual numbers, which we explain just below.

3.2 Dual Numbers

Dual numbers extend reals with a new element ε that has a property $\varepsilon^2 = 0$. A dual number can be written in a form $a+b\varepsilon$, where $a, b \in \mathbb{R}$. We denote the set of dual numbers as $\mathbb{R}_\varepsilon$. Dual numbers were proposed by the English mathematician W. Clifford in 1873 and later applied in geometry by the German mathematician E. Study. One of modern

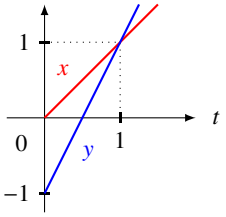

Fig. 3. Signals x and y for Example 8.

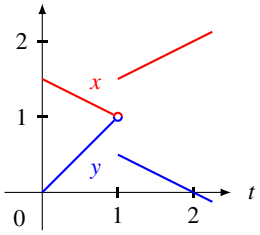

Fig. 4. Signals x and y for Examples 5 and 6.

applications of dual numbers and their extensions is in automatic differentiation [12]: one can exactly compute the value of the first derivative at a given point using the identity $f(x + \varepsilon) = f(x) + f'(x)\varepsilon$. Intuitively, ε can be understood as an infinitesimal value, and $a + b\varepsilon$ (for $b \neq 0$) is a point that is infinitely close to a. Polynomial functions can be extended to dual numbers, and via Taylor expansion, so can exponents, logarithms, and trigonometric functions. We work with piecewise-constant and piecewise-linear functions with real switching points, and we only make use of basic arithmetic. For example, if on the interval (b_1, b_2) the signal x is defined as $x(t) = a_1 t + a_0$, then $x(b_1 + \varepsilon) = a_1 b_1 + a_0 + a_1\varepsilon$ and $x(b_2 - \varepsilon) = a_1 b_2 + a_0 - a_1\varepsilon$.

Our primary use of a dual number is to represent a time point strictly after an event (a switching point, a threshold crossing, etc) but before any other event can happen, i.e., we use $t' + \varepsilon$ to represent the time point that happens right after t'. The coefficient 1 at ε denotes that time advances with the rate of 1 (although another consistently used coefficient works as well). Consequently, we also allow an output signal to produce a dual value, denoting a value that is attained at this dual time point. On the other hand, we require that signals are defined over real time, switching points of piecewise signals are reals, and time constants in formulas are reals. That is, dual-valued time is only used internally by the temporal operators and cannot be directly observed.

Minimum and Maximum of a Discontunuous Function. We also use dual-valued time to define the result of Min and Max for a discontinuous piecewise-linear function. The standard way to compute minimum and maximum of a continuous piecewise-linear function on a closed interval is based on the fact that they are attained at the endpoints of the interval or at the endpoints of the segments on which the function is defined. Using dual numbers, we extend it to dicontinuous functions: if for $t \in (b_1, b_2)$, $x(t) = a_1 t + a_2$ then we consider time points $b_1 + \varepsilon$ and $b_2 - \varepsilon$ as the candidates for reaching the minimum or maximum. Let us demonstrate this with an example.

Example 5 Consider the signal x defined as: "$x(t) = -0.5t + 1.5$ if $t \in [0, 1)$; $x(t) = 0.5t + 1$ if $t \geq 1$", as shown in Fig. 4. Let us find the minimum of x on the interval $[0, 2 + \varepsilon]$. By our definition, $\min_{t \in [0, 2+\varepsilon]} x(t) = \min\{x(0), x(1 - \varepsilon), x(1), x(2 + \varepsilon)\} = x(1 - \varepsilon) = 1 + 0.5\varepsilon$. This result should be understood as follows: $x(t)$ approaches the value of 1 from the above with derivative -0.5, but never reaches it.

Example 6 Our definition of minimum and maximum allows to correctly compare values of piecewise-linear functions around their discontinuity points. In Example 5, x never reaches the value of its lower bound, and our definition of minimum produces a dual

number that reflects this fact and also specifies the rate at which x appoaches its lower bound. This information would be lost if we computed the infimum of x. Again consider the signals in Fig. 4, with x defined as before, and "$y(t) = t,$ if $t \in [0, 1)$, $y(t) = -0.5t + 1,$ if $t \geq 1$". Let us evaluate at time $t = 0$ the formula $\mathrm{On}_{[0,2]}\,\mathrm{Min}\,x > \mathrm{On}_{[0,2]}\,\mathrm{Max}\,y$, which denotes the property $\forall t, t' \in [0, 2].\ x(t) > y(t')$. From the previous example, we have that $[\![\mathrm{On}_{[0,2]}\,\mathrm{Min}\,x]\!](0) = 1 + 0.5\varepsilon$. By a similar argument, $[\![\mathrm{On}_{[0,2]}\,\mathrm{Max}\,y]\!](0) = y(1 - \varepsilon) = 1 - \varepsilon$, which means that y approaches 1 from below with the rate of 1. Since, $1 + 0.5\varepsilon > 1 - \varepsilon$, our property holds at time 0, as expected.

We want to emphasize that while an output signal can take a dual value, its domain is considered to be a subset of reals. The semantics of temporal operators are allowed to internally use dual-valued time points, but has to produce an output signal that is defined over real time. This ensures that a piecewise signal always has real-valued switching points and that no event can happen at a dual-valued time point.

Example 7 Consider a formula $\varphi = \mathrm{F}_{[0,2]}(x = \mathrm{On}_{(-\inf,\inf)}\,\mathrm{Min}\,x)$, where x is as in Fig. 4. The meaning of φ is that within 2 time units x reaches its global minimum. In our semantics, this formula does not hold at time 0. By our definition, the global minimum of x is $1 + 0.5\varepsilon$, so the semantics of the formula at time 0 is equivalent to:

$$\begin{aligned}[\![\varphi]\!](0) &= [\![\mathrm{F}_{[0,2]}(x = 1 + 0.5\varepsilon)]\!](0)\\ &= \text{ if } \exists t \in \mathbb{T}.\ t \in [0, 2] \wedge x(t) = 1 + 0.5\varepsilon \text{ then 1 else 0}\end{aligned}$$

where $\mathbb{T} = [0, |w|] \subseteq \mathbb{R}$. There is no real value of time, where $x(t)$ yields a dual value, so the formula does not hold.

3.3 Semantics of Until

The On-operator allowed us to compute minima and maxima over a sliding window of fixed width. In this section, we introduce a new version of "until" operator that allows the window to have variable width that depends on the output signal of some formula.

Reinterpreting the classical Until as "Find First" Let us explain how we extend the "until" operator to work in the new setting. There already exists real-valued robust semantics of "until", but we do not believe it to be a good specification primitive. Instead, re-state standard the Boolean semantics and based on the re-stated version introduce the new real-(actually, dual-)valued semantics. Let us recall a possible semantics of untimed until in STL. Informally, "until" computes a conjunction of the values of the first operand over an interval that is not fixed, but defined by the second operand. Formally,

$$[\![p\ \mathrm{U}^{\mathrm{STL}}\ q]\!](t) = \exists t' \geq t.\ q(t') \wedge \forall s \in [t, t'].\ p(s)$$

To denote the STL version of "until" we write it with the superscript: $\mathrm{U}^{\mathrm{STL}}$, to distinguish from the new version that we define for our language. The version of "until" that we use in this paper is non-strict in the sense of [17]; it requites that p holds both at t and t'.

Efficient monitoring of STL "until" relies on instantiating the existential quantifier. The monitor scans the signal backwards and instantiates t' based on the earliest time point where q is true. The monitor needs to consider three cases shown in Fig. 5–7.

1. Fig. 5: q is false for every $t' \geq t$. Then the value of $p\ \mathrm{U}^{\mathrm{STL}}\ q$ at t is false.

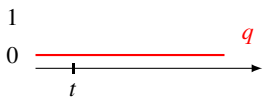

Fig. 5. Case 1: q is never true in the future.

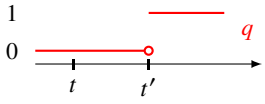

Fig. 6. Case 2: q there exists the earliest time point, where q becomes true.

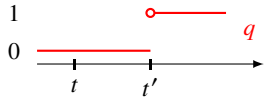

Fig. 7. Case 3: q becomes true, but there is no earliest time point.

2. Fig. 6: there exists the smallest $t' \geq t$, where q is true (this includes the case, where $t' = t$). Then the value of $p\ \mathrm{U}^{\mathrm{STL}}\ q$ at t is $\forall s \in [t, t']$. $p(s)$ (predicate p is not shown in the figure). The monitor needs not consider time points after t', since if "forall" produces false on a smaller interval, it will produce false on a larger one.
3. Fig. 7: q becomes true in the future, but there is no earliest time point. In this case, the monitor needs to take the universal quantification over an interval that ends just after t' (the switching point of q), but before any other event occurs. We can formalize this reasoning using dual numbers and say that the value of $p\ \mathrm{U}^{\mathrm{STL}}\ q$ at t is $\forall s \in [t, t' + \varepsilon]$. $p(s)$, where $t' + \varepsilon$ can be intuitively understood as a time point that happens after t', but before any other event can occur.

Below is the equivalent semantics of STL until that resolves the existential quantifier:

$$[\![p\ \mathrm{U}^{\mathrm{STL}}\ q]\!](t) = \begin{cases} \forall s \in [t, t']. \ p(s), \text{ if there exists the smallest } t' \geq t, \text{ s.t. } q(t') \\ \forall s \in [t, t' + \varepsilon]. \ p(s), \text{ where } t' = \inf\{t' | t' \geq t \wedge q(t')\}, \\ \qquad\qquad\qquad\qquad\qquad \text{if } \exists t' \geq t. \ q(t'), \text{ but there is no smallest } t' \\ \mathit{false}, \text{ otherwise} \end{cases}$$

Then, a monitor evaluates the universal quantifier via a finite conjunction, since in practice the signal p has finite variability, i.e. every interval is intersected by a finite number of constant segments.

Example 8 Let us consider two linear input signals: $x(t) = t$ and $y(t) = 2t - 1$ (see Fig. 3), and let us evaluate the formula $(y \leq x)\ \mathrm{U}^{\mathrm{STL}}\ (x > 1)$ at time 0 using non-strict "until" semantics. We define the earliest time point where $x > 1$ becomes true to be $1 + \varepsilon$, thus we need to evaluate the expression $\forall t \in [0, 1 + \varepsilon]$. $y(t) \leq x(t)$. At time $1 + \varepsilon$, we get $y(1 + \varepsilon) = 1 + 2\varepsilon > 1 + \varepsilon = x(1 + \varepsilon)$, thus the "until" formula does not hold. Informally, we can interpret the result as follows: when x becomes greater than 1, y becomes greater than x, while non-strict "until" requires that there exists a point, where both its left- and right-hand operands hold at the same time.

New Until as "Find First" At this point, extending "until" to produce a dual value is straightforward. With every time point, "until" possibly associates an interval, and we can compute an arbitrary aggregate function over it, instead of just conjunction. In fact, we introduce two flavors of "until". The first version: $\psi\ \mathrm{U}^{d}_{[a,b]}\ \varphi$ – works as follows. For every time point t, we either associate an interval ending when φ becomes non-zero (i.e., starts holding); or we report that no suitable end point was found. When such interval

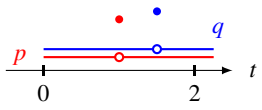

Fig. 8. Before time 2, an event p is followed by an event q.

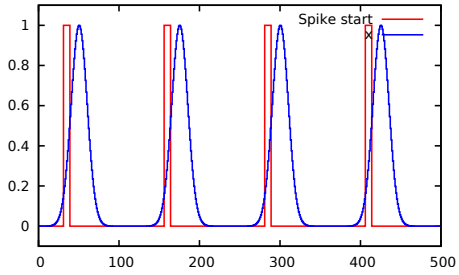

Fig. 9. A sequence of spikes and a Boolean signal marking the detected start times of spikes.

exists, we evaluate ψ on it. When the interval does not exist, we produce d. Formally,

$$[\![\psi \, \mathrm{U}^d_{[a,b]} \, \varphi]\!](t) = \begin{cases} [\![\psi]\!][t, t'], \text{ if } \exists \text{ the smallest } t' \in [t+a, t+b], \text{ s.t. } [\![\varphi]\!](t') \neq 0 \\ [\![\psi]\!][t, t'+\varepsilon], \text{ where } t' = \inf\{t' | t' \in [t+a, t+b] \wedge [\![\varphi]\!](t')\}, \\ \quad \text{if } \exists t' \in [t+a, t+b]. \, [\![\varphi]\!](t') \neq 0, \text{ but there is no smallest } t' \\ d, \text{ otherwise} \end{cases}$$

The second version: $\varphi_1 \downarrow \mathrm{U}^d_{[a,b]} \varphi_2$ does not perform aggregation, but evaluates φ_1 at the time point where φ_2 becomes non-zero, or produces d if such time point does not exist:

$$[\![\varphi_1 \downarrow \mathrm{U}^d_{[a,b]} \varphi_2]\!](t) = \begin{cases} [\![\varphi_1]\!](t'), \text{ if } \exists \text{ the smallest } t' \in [t+a, t+b], \text{ s.t. } [\![\varphi_2]\!](t') \neq 0 \\ [\![\varphi_1]\!](t'+\varepsilon), \text{ where } t' = \inf\{t' | t' \in [t+a, t+b] \wedge [\![\varphi_2]\!](t')\}, \\ \quad \text{if } \exists t' \in [t+a, t+b]. \, [\![\varphi_2]\!](t') \neq 0, \text{ but there is no smallest } t' \\ d, \text{ otherwise} \end{cases}$$

In a similar way, we could define past versions "until", where the interval $[a, b]$ refers to the past; we do not discuss them here due to space constraints.

STL Until The standard STL "until" can be expressed in the new language as follows:

$$\varphi_1 \, \mathrm{U}^{\mathrm{STL}}_{[a,b]} \, \varphi_2 = (\mathrm{Min} \, \varphi_1) \, \mathrm{U}^0_{[a,b]} \, \varphi_2$$

Lookup Using "until", we can express the "lookup" operator that queries the value of a signal at a point in the future, or returns some default value if the point does not exist.

$$\mathrm{D}^d_a \, \varphi = \varphi \downarrow \mathrm{U}^d_{[a,a]} 1$$

Example 9 (Spike) The ST-Lib library [14] uses the following formula to define a start point of a spike: $x' > m \wedge \mathrm{F}_{[0,d]}(x' < -m)$, where x' is the approximation of the right derivative $x'(t) = (x(t+\delta) - x(t))/\delta$, m is the magnitude of the spike, and d is the width. Using the lookup operator, we can include the definition of x' in the property itself:

$$(\mathrm{D}^y_\delta \, x - x)/\delta \geq m \wedge \mathrm{F}_{[0,d]}((\mathrm{D}^y_\delta \, x - x)/\delta \leq -m)$$

where y gives the value of the signal outside of its original domain.

Example 10 (Spike of Given Width and Height) Our language offers several alternative ways to define a spike. We can define a (start point of a) spike by composing two ramps: an increasing one, where the signal x increases by at least m withing w time units, and a decreasing one, where x decreases by at least m within w time units; the two ramps should be at most w units apart. The parameter w is the half-width of the spike.

$$(\mathrm{On}_{[0,w]}\,\mathrm{Max}\,x \geq x + m) \wedge \mathrm{F}_{[0,w]}(\mathrm{On}_{[0,w]}\,\mathrm{Min}\,x \leq x - m)$$

Fig. 9 shows an example of a series of spikes (blue) and a Boolean signal (red) that marks the detected start times of spikes.

Example 11 (TPTL-like Assertion) The second form of "until" allows to reason explicitly about time points and durations, somewhat similarly to TPTL. Consider the property "within 2 time units, we should observe an event p followed by an event q" (Fig. 8 shows an example of a satisfying signal). With some case analysis, this property can be expressed in MTL [5], but probably the best way to express it is offered by TPTL, that allows to assert "$c.\,\mathrm{F}(p \wedge \mathrm{F}(q \wedge c \leq 2))$", meaning "reset a clock c, eventually, we should observe p and from that point, eventually we should observe q, while the clock value will be at most 2". To express the property in our language, we introduce three auxiliary signals: $T(t) = t$ (which we use in some other examples as well), $pdelay = (T\downarrow \mathrm{U}^{\infty}p) - T$, which denotes the duration until the next occurrence of p and similarly $qdelay = (T \downarrow \mathrm{U}^{\infty}q) - T$, the duration until the next occurrence of q. Then, the property can be expressed as: $pdelay + (qdelay\downarrow \mathrm{U}^{\infty}p) \leq 2$

4 Monitoring

Similarly to other works on STL monitoring (e.g., [9]), we implement the algorithms for a subset of the language, and support the remaining operators via rewriting rules.

Rewriting of Until Similarly to STL, the timed "until" operator in our language can be expressed in terms of "eventually" (which is expressed using On) , "lookup", and untimed "until".

$$(\mathrm{Min}\,\varphi_1)\,\mathrm{U}^{d}_{[a,b]}\,\varphi_2 = \text{ if } \neg\,\mathrm{F}_{[a,b]}\,\varphi_2 \text{ then } d \text{ else } \mathrm{On}_{[0,a]}\,\mathrm{Min}((\mathrm{Min}\,\varphi_1)\,\mathrm{U}\,\varphi_2)$$

$$(\mathrm{Max}\,\varphi_1)\,\mathrm{U}^{d}_{[a,b]}\,\varphi_2 = \text{ if } \neg\,\mathrm{F}_{[a,b]}\,\varphi_2 \text{ then } d \text{ else } \mathrm{On}_{[0,a]}\,\mathrm{Max}((\mathrm{Max}\,\varphi_1)\,\mathrm{U}\,\varphi_2)$$

$$\varphi_1\downarrow \mathrm{U}^{d}_{[a,b]}\varphi_2 = \text{ if } \neg\,\mathrm{F}_{[a,b]}\,\varphi_2 \text{ then } d \text{ else } \mathrm{D}_a(\varphi_1\downarrow \mathrm{U}\varphi_2)$$

Let us prove that the first equivalence is true, and for the other two the proof idea is similar. Let t be the time point where we evaluate $(\mathrm{Min}\,\varphi_1)\,\mathrm{U}^{d}_{[a,b]}\,\varphi_2$ and its rewriting. If there is no time point $s \in [t + a, t + b]$ where φ_2 holds, both the original formula and its rewriting evaluate to d. Otherwise, let s be the earliest time point in $[t + a, t + b]$, where φ_2 holds, which can be a real or dual value, as explained in Section 3.3. Then the original formula evaluates to $\min\{[\![\varphi_1]\!](t') \mid t' \in [t, s]\}$. The rewritten formula at t evaluates to $\min\{[\![(\mathrm{Min}\,\varphi_1)\,\mathrm{U}\,\varphi_2]\!] \mid t' \in [t, t + a]\}$. Notice that for every t' there is a time point in the future, which we denote $g(t')$ where φ_2 holds, which is at most s, and for

$t' = t + a$ it is exactly s. That is, the rewritten formula evaluates to $\min\{\min\{[\![\varphi_1]\!](t'') \mid t'' \in [t', g(t')]\} \mid t' \in [t, t+a]\} = \min\{[\![\varphi_1]\!](t'') \mid t'' \in \bigcup\{[t', g(t')] \mid t' \in [t, t+a]\}\}$. Notice that since $g(t') \in [t', s]$ and $g(t+a) = s$, then $\bigcup\{[t', g(t')] \mid t' \in [t, t+a]\} = [t, s]$, and thus the rewritten formula evaluates to the same value as the original one.

Referring to Both Future and Past In the syntax, we allow the $\mathrm{On}_{[a,b]}$ operator to refer to both future and past, i.e, we allow the case when $a < 0$ and $b > 0$. Algorithms for Min/Max over a running window typically cannot work with this situation directly, and we need to apply the following rewriting: if $a < 0$ and $b > 0$,

$$\mathrm{On}_{[a,b]}\,\mathrm{Min}\,\varphi = \min\{\mathrm{On}_{[a,0]}\,\mathrm{Min}\,\varphi,\ \mathrm{On}_{[0,b]}\,\mathrm{Min}\,\varphi\}$$
$$\mathrm{On}_{[a,b]}\,\mathrm{Max}\,\varphi = \max\{\mathrm{On}_{[a,0]}\,\mathrm{Max}\,\varphi,\ \mathrm{On}_{[0,b]}\,\mathrm{Max}\,\varphi\}$$

Language of the Monitor The following subset of the language is equally expressive as the full language presented in (1). We implement the monitoring algorithms for this language, and the full syntax of (1) we support via rewriting.

$$\varphi ::= c \mid x \mid f(\varphi_1 \cdots \varphi_n) \mid \mathrm{On}_{[a,b]}\,\psi \mid \psi\ \mathrm{U}^d\ \varphi \mid \varphi_1 \downarrow \mathrm{U}^d \varphi_2 \mid \mathrm{D}^d_a\,\varphi$$
$$\psi ::= \mathrm{Min}\,\varphi \mid \mathrm{Max}\,\varphi$$

where either $a \geq 0$ or $b \leq 0$, i.e., the interval $[a, b]$ cannot refer to both future and past.

All operators in the language of the monitor admit efficient offline monitoring. Minimum and maximum over a sliding window required by the On-operator can be computed using a variation of D. Lemire's algorithm [15,9]; "lookup" operator D shifts its input signal by a constant distance; and for untimed "until" we can scan the input signal backwards and perform a special case of running minimum or maximum.

4.1 Monitoring Algorithms

In this section, we briefly describe monitoring algorithms for piecewise-constant signals.

Representation of Signals We represent a piecewise-constant function $\mathbb{T} \to \mathbb{R}$ or $\mathbb{T} \to \mathbb{R}_\varepsilon$ as a sequence of segments: $\langle s_0, s_1, \ldots, s_{m-1}\rangle$, where every segment $s_i = J_i \mapsto v_i$ maps an interval J_i to a real or dual value v_i. The intervals J_i form a partition the domain of the signal and are ordered in ascending time order, i.e., $\sup J_i = \inf J_{i+1}$ and $J_i \cap J_{i+1} = \varnothing$. The domain of the signal signal corresponding to the sequence $u = \langle J_0 \mapsto v_o, \ldots, J_{m-1} \mapsto v_{m-1}\rangle$ is denoted by $dom(u) = J_0 \cup \ldots \cup J_{m-1}$. For example, if the function $x(t)$ is defined as $x(t) = 0$, if $t \in [0, 1)$, and $x(t) = 1$, if $t \in [1, 2]$, then $x(t)$ is represented by the sequence $u_x = \langle [0, 1) \mapsto 0, [1, 2] \mapsto 1\rangle$, and $dom(u_x) = [0, 2]$.

Empty brackets $\langle\rangle$ denote an empty sequence that does not represent a valid signal, but can be used by algorithms as an intermediate value. We manipulate the sequences with two main operations. The function *append* adds a segment to the end of a sequence: $append(\langle s_0, \ldots, s_{m-1}\rangle, s') = \langle s_0, \ldots, s_{m-1}, s'\rangle$. The function *prepend* adds a segment to the start of a sequence: $prepend(\langle s_0, \ldots, s_{m-1}\rangle, s') = \langle s', s_0, \ldots, s_{m-1}\rangle$. This may produce a sequence where the first segment does not start time at time 0. While such a sequence does not represent a valid signal, it can be used by the algorithms as an intermediate value. The function *removeLast* removes the last segment of a sequence, assuming it was non-empty: $removeLast(\langle s_0, \ldots, s_{m-1}\rangle) = \langle s_0, \ldots, s_{m-2}\rangle$.

```
function until(u_1, u_2, f, d)
  let u_1 = ⟨J^1_0 ↦ v^1_0, ..., J^1_{m-1} ↦ v^1_{m-1}⟩
  let u_2 = ⟨J^2_0 ↦ v^2_0, ..., J^2_{k-1} ↦ v^2_{k-1}⟩
  i ← m − 1, j ← k − 1
  (u_r, s, v′) ← (⟨⟩, 0, d)
  while i ≥ 0 ∧ j ≥ 0 do
    J ← J^1_i ∩ J^2_j
    (u_r, s, v′) ← untilAdd(u_r, s, v′, J, v^1_i, v^2_j)
    if ∃t_1 ∈ J^1_i. ∀t_2 ∈ J^2_j. t_1 > t_2 then
      j ← j + 1
    else if ∃t_2 ∈ J^2_j. ∀t_1 ∈ J^1_i. t_2 > t_1
    then
      i ← i + 1
    else
      i ← i + 1, j ← j + 1
    end
  end
  return u_r
end
```

```
function untilAdd(u_r, s, v′, J, v_1, v_2)
  if v_2 ≠ 0 then
    v′ ← v_1
    s ← 1
  else if s ≠ 0 then
    v′ ← f(v′, v_1)
  end
  prepend(u_r, J ↦ v′)
  return (u_r, s, v′)
end
```

Fig. 10. Algorithm for monitoring "until"-formulas.

An output signal of a formula is scalar-valued and is represented by one such sequence. An input signal usually has multiple components, i.e., it is a function $\mathbb{T} \to \mathbb{R}^n$, and is represented by a set of n sequences.

On-Formulas For $\mathrm{On}_{[a,b]}\,\mathrm{Min}\,\varphi$ and $\mathrm{On}_{[a,b]}\,\mathrm{Max}\,\varphi$, a monitor needs to compute the minimum or maximum of the output signal of φ over the sliding window. The corresponding algorithm was developed for discrete time by D. Lemire [15] and later adapted for continuous time [9].

Lookup-Formulas Computing the output signal for $\mathrm{D}_a^d\,\varphi$ is straightforward. We need to shift every segment of u_φ (the representation of the output signal of φ) to the left by a truncating at 0 and append a padding segment with the value of d.

Until-Formulas Informally, monitoring the "until"-formulas, $\mathrm{Min}\,\varphi_1\,\mathrm{U}^d\,\varphi_2$, $\mathrm{Max}\,\varphi_1\,\mathrm{U}^d\,\varphi_2$, and $\varphi_1\downarrow\,\mathrm{U}^d\varphi_2$, works as follows. The monitor scans the output signals of φ_1 and φ_2 backwards. While φ_2 evaluates to a non-zero value, the monitor outputs the value of φ_1. When φ_2 evaluates to 0, the monitor outputs either the default value (if the monitor did not yet encounter a non-zero value of φ_2), or the running minimum or maximum of φ_1, or the value that φ_1 had at the last time point where φ_2 was non-zero.

The function *until* and *untilAnd* in Fig. 10 implement this idea. The inputs to the function *until* are: sequences u_1 and u_2 representing the output signals of φ_1 and φ_2 (with $dom(u_1) = dom(u_2)$), default value d, and the function f used for aggregation; it can be min, max, or the special function $\lambda x, y.\ x$ which returns the value of its first argument and which we use to monitor the formula $\varphi_1\downarrow\,\mathrm{U}^d\varphi_2$. The function *until* scans the input

sequences backwards and iterates over intervals where both input signals maintain a constant value (J) . Each such interval is passed to the function *untilAdd*, which updates the state of the algorithm (v', s) and constructs the output signal (u_r).

5 Implementation and Experiments

We implemented the monitoring algorithm in a prototype tool that is available at `https://gitlab.com/abakhirkin/StlEval`. The tool has a number of limitations, notably it can only use piecewise-constant interpolation (so we cannot evaluate examples that use the auxiliary signal $T(t) = t$) and does not support past-time operators. It is written in C++ and uses double-precision floating point numbers for time points and signal values. We evaluate the tool using a number of synthetic signals and a number of properties based on the ones described earlier in the paper.

Signals We use the following signals discretized with time step 1.

- x_{sin} – sine wave with amplitude 1 and period 250; see red curve in Fig. 2.
- x_{decay} – damped oscillation with period 250. For $t \in [0, 1000)$, x defined as $x_{\text{decay}}(t) = \frac{1}{e}\sin(250t + 250)e^{-\frac{1}{250}x}$, see red curve in Fig. 1; for $t \geq 1000$, the pattern repeats;
- x_{spike} – series of spikes; a single spike is defined for $t \in [0, 125)$ as: $x_{\text{spike}}(t) = e^{\frac{(t-50)^2}{2\cdot 10^2}}$, and after that the pattern repeats; see blue curve in Fig. 9.

Properties We use the following properties:

- $\varphi_{\text{stab}} = \text{G F}\left(\text{On}_{[0,200]}\,\text{Max}\,x - \text{On}_{[0,200]}\,\text{Min}\,x \leq 0.1\right)$, x always eventually becomes stable around some value for 200 time units.
- $\varphi_{\text{stab}-0} = \text{G F G}_{[0,200]}(|x| \leq 0.05)$: x always eventually becomes stable around 0 for 200 time units.
- $\varphi_{\text{until}} = \text{G}_{[0,20k]}\,\text{F}\left((\text{Max}\,x)\,\text{U}^{\infty}_{[200,\infty)}\,(|x'| \geq 0.1)\right) - \left((\text{Min}\,x)\,\text{U}^{-\infty}_{[200,\infty)}\,(|x'| \geq 0.1)\right) \leq 0.1$, where $x' = (\text{D}^0_1\,x - x)$, x always eventually becomes stable for at least 200 time units and then starts changing with derivative of at least 0.1.
- $\varphi_{\text{max}-\text{min}} = \text{G}\left((x \geq \text{On}_{[0,85]}\,\text{Max}\,x) \Rightarrow \text{F}(x \leq \text{On}_{[0,85]}\,\text{Min}\,x)\right)$, every local maximum is followed by a local minimum.
- $\varphi_{\text{above}-\text{below}} = \text{G}\left(x \geq 0.85 \Rightarrow (\text{F}\,x \leq -0.85)\right)$, if x is above 0.85, it should eventually become below −0.85.
- $\varphi_{\text{spike}} = (\text{On}_{[0,16]}\,\text{Max}\,x \geq x + 0.5) \wedge \text{F}_{[0,16]}(\text{On}_{[0,16]}\,\text{Min}\,x \leq x - 0.5)$, spike of half-width 16 and height at least 0.5.
- $\varphi_{\text{spike}-\text{stlib}} = \text{F}\left(x' \geq 0.04 \wedge \text{F}_{[0,25]}(x' \leq -0.04)\right)$, where $x' = (\text{D}^0_1\,x - x)$, spike of width at most 25 and magnitude 0.04.

Some properties are expressed in our language using On- and "until"-operators, and some are STL properties. This allows us to see how much time it takes to monitor a more complicated property in our language (e.g., φ_{stab}, stabilization around an unknown value) compared to a similar but more simple STL property (e.g., $\varphi_{\text{stab}-0}$, stabilization around a known value). In our experiments we see a constant factor between 2 and 5.

Table 1 shows the evaluation results. A row gives a formula and a signal shape; a column gives the number of samples in the input signal, and a table cell gives two time figures in seconds: the monitoring time excluding the time required to read the input

Table 1. Monitoring time for different formulas and signals.

		This paper				AMT 2.0				Breach			
		100k		1M		100k		1M		100k		1M	
φ_{stab}	x_{decay}	0.004	0.05	0.048	0.39								
$\varphi_{\text{stab}-0}$	x_{decay}	0.003	0.04	0.023	0.38	0.59	4.0	2.4	7.3	0.053	-	0.42	-
φ_{until}	x_{decay}	0.01	0.05	0.097	0.43								
$\varphi_{\text{max}-\text{min}}$	x_{sin}	0.007	0.04	0.07	0.4								
$\varphi_{\text{above}-\text{below}}$	x_{sin}	0.002	0.04	0.02	0.36	0.6	3.1	2.4	7.5	0.05	-	0.4	-
φ_{spike}	x_{spike}	0.01	0.05	0.1	0.45								
$\varphi_{\text{spike}-\text{stlib}}$	x_{spike}	0.006	0.05	0.05	0.43	1.0	4.0	5.0	13	0.058	-	0.47	-

data, and the total runtime of an executable. We note that for our tool, the total runtime is dominated by the time required to read the input signal from a text file. For the three STL properties we include the time it took AMT 2.0 (a monitoring tool written in Java [18]) and Breach (a Matlab toolbox partially written in C++ [8]; Breach does not have a standalone executable, so the we leave the corresponding columns empty) to evaluate the formula. This way we show that our implementation of STL monitoring has good enough performance to be used as a baseline when evaluating the cost of the added expressiveness in the new language. Time figures were obtained using a PC with a Core i3-2120 CPU and 8GB RAM running 64-bit Debian 8.

6 Conclusion and Future Work

We describe a new specification language that extends STL with the ability to produce and manipulate real-valued output signals (while in STL, every formula has a Boolean output signal). Properties in the new language are specified in terms of minima and maxima over a sliding window, which can have fixed width, when using a generalization of F- and G-operators, or variable width, when using a new version "until". We show how the new language can express properties that motivated the creation of more expressive and harder to monitor logics. Offline monitoring for the new language is almost as efficient as STL monitoring; the complexity is linear in the length of the input signal and does not depend on the constants appearing in the formula.

There are multiple directions for future work; perhaps more interesting one is adding integration over a sliding window (in addition to minimum and maximum). This is already allowed by some formalisms [7], and when added to our language will allow to assert that a signal approximates the behaviour of a system defined by a given differential equation (since we will be able to assert $y(t) \approx \int_0^t x(t)dt$). Before making integration available, we wish to investigate how to better deal in a specification language with approximation errors. Finally, we wish to make our language usable in falsification, which means that for every formula with Boolean output signal we wish to be able to compute a real-valued robustness measure.

Acknowledgements The authors thank T. Ferrére, D. Nickovic, E. Asarin for comments on the draft of this paper, and O. Lebeltel for providing a version of AMT for the experiments.

References

1. Alur, R., Henzinger, T.A.: A really temporal logic. J. ACM 41(1), 181–204 (1994)
2. Bakhirkin, A., Ferrère, T., Henzinger, T.A., Nickovic, D.: The first-order logic of signals: keynote. In: Brandenburg, B.B., Sankaranarayanan, S. (eds.) International Conference on Embedded Software (EMSOFT). pp. 1:1–1:10. ACM (2018)
3. Bartocci, E., Deshmukh, J.V., Donzé, A., Fainekos, G.E., Maler, O., Nickovic, D., Sankaranarayanan, S.: Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification - Introductory and Advanced Topics, LNCS, vol. 10457, pp. 135–175. Springer (2018)
4. Basin, D.A., Klaedtke, F., Müller, S., Zalinescu, E.: Monitoring metric first-order temporal properties. J. ACM 62(2), 15:1–15:45 (2015)
5. Bouyer, P., Chevalier, F., Markey, N.: On the expressiveness of TPTL and MTL. Inf. Comput. 208(2), 97–116 (2010)
6. Brim, L., Dluhos, P., Safránek, D., Vejpustek, T.: Stl*: Extending signal temporal logic with signal-value freezing operator. Inf. Comput. 236, 52–67 (2014)
7. Claessen, K., Smallbone, N., Eddeland, J., Ramezani, Z., Akesson, K.: Using valued booleans to find simpler counterexamples in random testing of cyber-physical systems. In: Workshop on Discrete Event Systems (WODES) (2018)
8. Donzé, A.: Breach, A toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P.B. (eds.) Computer Aided Verification (CAV). LNCS, vol. 6174, pp. 167–170. Springer (2010)
9. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for STL. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification (CAV). LNCS, vol. 8044, pp. 264–279. Springer (2013)
10. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) Formal Modeling and Analysis of Timed Systems (FORMATS). LNCS, vol. 6246, pp. 92–106. Springer (2010)
11. Elgyütt, A., Ferrère, T., Henzinger, T.A.: Monitoring temporal logic with clock variables. In: Jansen, D.N., Prabhakar, P. (eds.) Formal Modeling and Analysis of Timed Systems (FORMATS). LNCS, vol. 11022, pp. 53–70. Springer (2018)
12. Fike, J.A., Alonso, J.J.: Automatic differentiation through the use of hyper-dual numbers for second derivatives. In: Forth, S., Hovland, P., Phipps, E., Utke, J., Walther, A. (eds.) Recent Advances in Algorithmic Differentiation. pp. 163–173. Springer, Berlin, Heidelberg (2012)
13. Havelund, K., Peled, D.: Efficient runtime verification of first-order temporal properties. In: Gallardo, M., Merino, P. (eds.) International Symposium on Model Checking Software (SPIN). LNCS, vol. 10869, pp. 26–47. Springer (2018)
14. Kapinski, J., Jin, X., Deshmukh, J., Donze, A., Yamaguchi, T., Ito, H., Kaga, T., Kobuna, S., Seshia, S.: St-lib: A library for specifying and classifying model behaviors. In: SAE Technical Paper. SAE International (04 2016)
15. Lemire, D.: Streaming maximum-minimum filter using no more than three comparisons per element. Nordic Journal of Computing 13(4) (2006)
16. 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 and Formal Techniques in Real-Time and Fault-Tolerant Systems (FORMATS/FTRTFT). LNCS, vol. 3253, pp. 152–166. Springer (2004)
17. Nickovic, D.: Checking Timed and Hybrid Properties: Theory and Applications. (Vérification de propriétés temporisées et hybrides: théorie et applications). Ph.D. thesis, Joseph Fourier University, Grenoble, France (2008)

18. Nickovic, D., Lebeltel, O., Maler, O., Ferrère, T., Ulus, D.: AMT 2.0: Qualitative and quantitative trace analysis with extended signal temporal logic. In: Beyer, D., Huisman, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems (TACAS). LNCS, vol. 10806, pp. 303–319. Springer (2018)
19. Rodionova, A., Bartocci, E., Nickovic, D., Grosu, R.: Temporal logic as filtering. In: Dependable Software Systems Engineering, pp. 164–185 (2017)
20. Silvetti, S., Nenzi, L., Bartocci, E., Bortolussi, L.: Signal convolution logic. In: Lahiri, S.K., Wang, C. (eds.) Automated Technology for Verification and Analysis (ATVA). LNCS, vol. 11138, pp. 267–283. Springer (2018)