Efficient Robust Monitoring for STL

Alexandre Donzé[1], Thomas Ferrère[2], Oded Maler[2]

[1]University of California, Berkeley, EECS dept, `donze@eecs.berkeley.edu`
[2] Verimag, CNRS and Grenoble University, {`maler`|`ferrere`}`@imag.fr`

Abstract. Monitoring transient behaviors of real-time systems plays an important role in model-based systems design. Signal Temporal Logic (STL) emerges as a convenient and powerful formalism for continuous and hybrid systems. In this paper we present an efficient algorithm for computing the robustness degree in which a piecewise-continuous signal satisfies or violates an STL formula. Our algorithm, by leveraging state-of-the-art streaming algorithms from Signal Processing, is linear in the size of the signal and its implementation in the Breach tool is shown to outperform alternative implementations.

1 Introduction

Temporal Logic (TL) is a popular formalism, introduced into systems design [Pnu77] as a language for specifying acceptable behaviors of reactive systems. Traditionally, it has been used for formal verification, either by deductive methods [MP91,MP95], or algorithmic methods (model checking [CGP99,QS82]). In this framework, the behaviors in question are typically discrete, that is, sequences of states and/or events. Two other assumptions concerning this use of TL in verification are implicit:

1. Systems correctness is affirmed if *all* system behaviors satisfy the specification. Thus model checking is based on composing the system model with an automaton for the specification and analyzing all possible paths in the combined transition system;
2. The satisfaction of a property by a behavior is a purely discrete matter (yes/no), which is in the spirit of most logics.[1]

In recent years, several trends suggest alternative ways to use TL in the *design* of complex systems and also during their *operations*. The first trend is due to the state-explosion wall, which limits the size of systems that can be verified (not to mention systems like programs with numerical variables or hybrid systems where verification is not even decidable). As a result we can see a proliferation of statistical methods a-la Monte-Carlo, where universal quantification is replaced by random simulation, with and even without statistical coverage guarantees. In this framework, also known as *runtime verification*, *assertion checking* or

[1] There are some branches of multi-valued logic such as Fuzzy [Zad65] and probabilistic [Nil86] but mainstream Logic is about *true* and *false*.

monitoring, the temporal formula is still used for a rigorous specification of the requirements, but unlike model-checking, it is evaluated on a *single* behavior at a time, a much easier task.

Unlike formal verification, monitoring does *not* require a *model* of the system. All it needs is a process that generates observable behaviors. As such it can be applied to systems which are viewed as *black boxes* either because their developers want to protect their intellectual property or because it is a complex program without a decent and tractable formal model. For the same reason, temporal property checking can be integrated in monitoring and diagnostics of *real* systems during their operation and provide more refined means to define and detect hazardous situations.

The present paper is based on *signal temporal logic* (STL) , a formalism for specifying properties of dense-time quantitative signals [MN04,MNP08], for which a monitoring tool called AMT [NM07] has been developed and used in the context of analog and mixed-signal circuits [JKN10,MN12]. In many real-life applications, especially when dealing with continuous dynamics and numerical quantities, yes/no answers provide only partial information and could be augmented with *quantitative* information about the satisfaction to provide a better basis for decision making. To illustrate, consider the formula $x < c$ for constant c and a real-valued variable x ranging over some domain X. The formula splits X into $X^0 = \{x : x \geq c\}$ and $X^1 = \{x : x < c\}$. The latter is called the *validity domain* of the formula. When we pick a number $x \in X$, the answer to the satisfaction query $x \models \mathbf{x} > c$ depends on the *membership* of x in X^1 but not on its relative position inside or outside X^1. The *robustness degree* of the satisfaction should tell us whether x satisfies the formula by far ($x \ll c$) or very marginally ($x = c - \epsilon$ for a small positive ϵ). For this example, the robustness degree is captured by $c - x$ whose *sign* indicates satisfaction/violation and its *magnitude* indicates the *distance* between x and the *boundary* between X^0 and X^1.

Such notions have been introduced into TL by Fainekos and Pappas [FP09] for STL and by Fages and Rizk [RBFS08] for LTL over real-valued sequences. The robustness information is useful to assess the severity of a detected malfunctioning of a working systems. It can also increase the confidence in the results of incomplete-coverage validation techniques, if it so happens that all sampled behaviors satisfy the requirements robustly. In a previous paper [DM10] we have introduced notions of robustness both in space and time, and provided an algorithm for computing the robustness degree with respect to a given signal. Signals are represented as sequences of time-stamped points and are interpreted as piecewise-linear via interpolation.

The major contribution of this paper is a new optimal algorithm that computes the robustness degree for such a signal in time linear with respect to the size of the signal (number of points). This algorithm guarantees that the overhead added by monitoring to the simulation process is acceptable, thus making robustness-based monitoring a feasible technology that can be used routinely as an add-on for simulation engines. This low complexity is due to two key ideas:

- The use of the optimal streaming algorithm of Daniel Lemire [Lem06] to compute the min and max of a numeric sequence over a sliding window;
- The rewriting of the (bounded) timed "until" operator [DT04] as a conjunction of simpler timed and untimed operators.

The algorithm has been implemented at the core of Breach [Don10] which is a highly versatile toolbox for simulation-based analysis of complex systems, recently applied to biological reaction networks [DFG$^+$11,MDMF12], to mine requirements of Simulink models in the automative industry [JDDS13] and to characterize patterns in musical signals [DMB$^+$12]. Our implementation outperforms the tool S-TaLiRo [ALFS11], to the best of our knowledge the only other tool implementing quantitative semantics for dense time.

The rest of the paper is organized as follows. In Section 2, we recall the main definitions of STL and its quantitative semantics. In Section 4 we present our robustness computation framework and describe the algorithms for simple operators, such as negation and conjunction. Section 4 treats in details the case of untimed *until* and timed *eventually*, which completes the algorithms presentation. Section 5 discusses the theoretical worst-case complexity of the computation and Section 6 provides experimental results.

2 Signal Temporal Logic

In this section we recall the framework set in [MN04] to specify properties of real-valued signals, we extend it to a multi-valued logic as proposed by [FP09], and present the main properties of this extension.

We adopt the following conventions. The set of Boolean values is taken as $\mathbb{B} := \{\bot, \top\}$, with $\bot < \top$, $-\top = \bot$ and $-\bot = \top$, inducing the well known algebra. We write $\overline{\mathbb{R}} := \mathbb{R} \cup \mathbb{B}$ for the totally ordered set of real numbers with smallest element $\bot$ and greatest element $\top$.

A *signal* will be a function $D \to E$, with D an interval of $\mathbb{R}^+$ and $E \subset \overline{\mathbb{R}}$. Signals with $E = \mathbb{B}$ are called *Boolean* signals, whereas those where $E = \mathbb{R}$ are *real-valued* signals. An execution *trace* w is a set of real-valued signals $\{x_1^w, ..., x_k^w\}$ defined over some interval D of $\mathbb{R}^+$, which is called the *time domain* of w. Such a trace can be "booleanized" through a set of threshold predicates of the form $x_i \geq 0$. Signal Temporal Logic is then a simple extension of Metric Temporal Logic where real-valued variables $(x_i)_{i \in \mathbb{N}}$ are transformed into Boolean values via these predicates[2] The syntax of STL will be taken as follows:

$$\varphi := \text{true} \mid x_i \geq 0 \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \, \mathbf{U}_I \, \varphi$$

Here x_i are variables, and I is a closed, non-singular interval of $\mathbb{R}^+$. This includes bounded intervals $[a, b]$ and unbounded intervals $[a, +\infty)$ for any $0 \leq a < b$.

Let w be a trace of time domain D. The formula φ is said to be *defined* over a time interval $\text{dom}(\varphi, w)$ given by the following rules: $\text{dom}(\text{true}, w) =$

[2] More expressive predicates could be added in the form of a preprocessing step, which we do not include explicitely in our framework.

$\mathrm{dom}(x_i \geq 0, w) = D$, $\mathrm{dom}(\neg\varphi, w) = \mathrm{dom}(\varphi, w)$, $\mathrm{dom}(\varphi \wedge \psi, w) = \mathrm{dom}(\varphi, w) \cap \mathrm{dom}(\psi, w)$, $\mathrm{dom}(\varphi \,\mathbf{U}_I\, \psi, w) = \{t \in \mathbb{R} \mid t + [0, \inf(I)] \subset \mathrm{dom}(\varphi, w) \text{ and } t + \inf(I) \in \mathrm{dom}(\psi, w)\}$.

Boolean Semantics For a trace w, the validity of an STL formula φ at a given time $t \in \mathrm{dom}(\varphi, w)$ is set according to the following inductive definition.

$$\begin{array}{lll}
w, t \vDash \text{true} & & \\
w, t \vDash x_i \geq 0 & \text{iff} & x_i^w(t) \geq 0 \\
w, t \vDash \neg\varphi & \text{iff} & w, t \nvDash \varphi \\
w, t \vDash \varphi \wedge \psi & \text{iff} & w, t \vDash \varphi \text{ and } w, t \vDash \psi \\
w, t \vDash \varphi \,\mathbf{U}_I\, \psi & \text{iff} & \text{exists } t' \in t + I \text{ s.t. } w, t' \vDash \psi \text{ and for all } t'' \in [t, t'], w, t'' \vDash \varphi
\end{array}$$

We can redefine other usual operators as syntactic abbreviations:

$$\begin{array}{ll}
\text{false} := \neg\text{true} & \varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi) \\
\Diamond_I\, \varphi := \text{true}\, \mathbf{U}_I\, \varphi & \Box_I\, \varphi := \neg\Diamond_I\, \neg\varphi
\end{array}$$

We use $\Diamond$ and $\mathbf{U}$ as shorthands for *untimed* operators $\Diamond_{[0,+\infty)}$ and $\mathbf{U}_{[0,+\infty)}$. For a given formula φ and execution trace w, we define the *satisfaction signal* $\chi(\varphi, w, .)$ as follows:

$$\text{for all } t \in \mathrm{dom}(\varphi, w) \quad \chi(\varphi, w, t) := \begin{cases} \top \text{ if } w, t \vDash \varphi \\ \bot \text{ otherwise} \end{cases}$$

Monitoring the satisfaction of a formula φ can be done by computing for each subformula ψ of φ the entire satisfaction signal $\chi(\psi, w, .)$. The procedure is recursive on the structure of the formula, and goes from the atomic predicates up to the top formula [MN04].

Quantitative Semantics Given a formula φ, trace w, and time $t \in \mathrm{dom}(\varphi, w)$, we define the quantitative semantics $\rho(\varphi, w, t)$ by induction as follows:

$$\begin{aligned}
\rho(\text{true}, w, t) &= \top \\
\rho(x_i \geq 0, w, t) &= x_i^w(t) \\
\rho(\neg\varphi, w, t) &= -\rho(\varphi, w, t) \\
\rho(\varphi \wedge \psi, w, t) &= \min\{\rho(\varphi, w, t), \rho(\psi, w, t)\} \\
\rho(\varphi \,\mathbf{U}_I\, \psi, w, t) &= \sup_{t' \in t+I} \min\{\rho(\psi, w, t'), \inf_{t'' \in [t,t']} \rho(\varphi, w, t'')\}
\end{aligned}$$

It is worth noting that if we let $\chi(x_i \geq 0, w, t) = \begin{cases} \top \text{ if } x_i^w(t) \geq 0 \\ \bot \text{ otherwise} \end{cases}$ and apply to χ the above inductive rules, we fall back to Boolean signals and obtain an equivalent characterization of χ. In the quantitative semantics however, atomic predicates $x_i \geq 0$ do not evaluate to $\top$ or $\bot$ but give a real value representing the distance to satisfaction or to violation, which is then propagated in the formula using the $\{\min, \max, -\}$ operations on $\overline{\mathbb{R}}$.

From the lattice properties of $(\overline{\mathbb{R}}, <)$, we are granted the axioms of associativity, commutativity, neutral element, and distributivity. The minus function remains involutive, which gives us the usual de Morgan laws $\neg(\varphi \vee \psi) \sim \neg\varphi \wedge \neg\psi$ and $\neg \Diamond_I \varphi \sim \Box_I \neg\varphi$. Derived operators enjoy the same natural interpretation as in the Boolean semantics: $\rho(\varphi \vee \psi, w, t) = \max\{\rho(\varphi, w, t), \rho(\psi, w, t)\}$, $\rho(\Diamond_I \varphi, w, t) = \sup\limits_{t' \in t+I} \rho(\varphi, w, t')$, and $\rho(\Box_I \varphi, w, t) = \inf\limits_{t' \in t+I} \rho(\varphi, w, t')$.

Property of Robustness Estimate The quantitative semantics of STL have two fundamental properties, that would alone justify their introduction. Firstly, whenever $\rho(\varphi, w, t) \neq 0$ its sign indicates the satisfaction status.

Theorem 1 (Soundness). *Let φ be an STL formula, w a trace and t a time.*

$$\rho(\varphi, w, t) > 0 \Rightarrow w, t \vDash \varphi$$
$$\rho(\varphi, w, t) < 0 \Rightarrow w, t \nvDash \varphi$$

Secondly, if w satisfies φ at time t, any other trace w' whose pointwise distance from w is smaller than $\rho(\varphi, w, t)$ also satisfies φ at time t.

Theorem 2 (Correctness). *Let φ be an STL formula, w and w' traces over the same time domain, and $t \in \mathrm{dom}(\varphi, w)$.*

$$w, t \vDash \varphi \text{ and } \|w - w'\|_\infty < \rho(\varphi, w, t) \quad \Rightarrow \quad w', t \vDash \varphi$$

On these grounds we now talk of ρ as the *robustness estimate*. For a given trace w, and φ an STL formula, we will refer to the *robustness signal* of φ with respect to w, as the signal $\rho(\varphi, w, .)$. Similarly to the satisfaction signal, it is defined over the time domain $\mathrm{dom}(\varphi, w)$.

Until Rewrite The properties $\varphi \, \mathbf{U}_{[a,b]} \, \psi \sim \Diamond_{[a,b]} \psi \wedge \varphi \, \mathbf{U}_{[a,+\infty)} \, \psi$ and $\varphi \, \mathbf{U}_{[a,+\infty)} \, \psi \sim \Box_{[0,a]} (\varphi \, \mathbf{U} \, \psi)$ extend from Boolean to quantitative semantics.

Lemma 1. *For φ, ψ two STL formula, w a trace and any time t where defined,*

$$\rho(\varphi \, \mathbf{U}_{[a,b]} \, \psi, w, t) = \rho(\Diamond_{[a,b]} \psi \wedge \varphi \, \mathbf{U}_{[a,+\infty)} \, \psi, w, t) \tag{1}$$
$$\rho(\varphi \, \mathbf{U}_{[a,+\infty)} \, \psi, w, t) = \rho\big(\Box_{[0,a]} (\varphi \, \mathbf{U} \, \psi), w, t\big) \tag{2}$$

Proof. We only prove the first rewrite rule, the second can be obtained by a similar argument. Let us take y, y' the robustness signals of φ, ψ relatively to w. We note $u := \sup\limits_{\tau \in [a,b]} \min\{y'(\tau), \inf\limits_{[0,\tau]} y\}$ and $v := \min\big\{\sup\limits_{[a,b]} y', \sup\limits_{\tau \in [a,+\infty)} \min\{y'(\tau), \inf\limits_{[0,\tau]} y\}\big\}$ the robustness values of the main formulas of (1) at a given time t. Suppose that $u \neq v$, for instance $u < v$. We define the signals $x : t \mapsto y(t) - \frac{u+v}{2}$ and $x' : t \mapsto y'(t) - \frac{u+v}{2}$. Now consider the formulas $\gamma := (x \geq 0) \, \mathbf{U}_{[a,b]} \, (x' \geq 0)$ and $\theta := \Diamond_{[a,b]} (x' \geq 0) \wedge (x \geq 0) \, \mathbf{U}_{[a,+\infty)} \, (x' \geq 0)$. By commutation of constants with sup and inf we get $\rho(\gamma, w, t) = u - \frac{u+v}{2} < 0$ and $\rho(\theta, w, t) = v - \frac{u+v}{2} > 0$, so that $w, t \nvDash \gamma$ while $w, t \vDash \theta$. This is impossible, as $\gamma \sim \theta$ in the Boolean semantics.

3 Computing the Robustness Estimate

While monitoring a system, real or simulated, signals are available to us as finite timed words over the alphabet $\mathbb{R}^n$. We interpret these by linear interpolation. This section presents the basic framework for computing robustness under this hypothesis. The general robustness computation procedure is as follows.

Algorithm 1 Robustness(φ, w)

```
switch ($\varphi$)
case true:
    return $\overline{\top}$    % a constant $\top$ signal
case $x_i \geq 0$:
    return $x_i^w$
case $* \ \varphi_1$:
    $y :=$ Robustness($\varphi_1, w$)
    return Compute($*, y$)
case $\varphi_1 \ * \ \varphi_2$:
    $y :=$ Robustness($\varphi_1, w$)
    $y' :=$ Robustness($\varphi_2, w$)
    return Compute($*, y, y'$)
end switch
```

Definition 1. *A signal y is said to be finitely piecewise-linear, continuous (*f.p.l.c. *for short) if there exists a finite sequence $(t_i)_{i \leq n_y}$ such that:*

- *the definition domain of y is $[t_0, t_{n_y})$*
- *for all $i < n_y$, $\quad y$ is continuous at t_i and affine on $[t_i, t_{i+1})$*

$(t_i)_{i \leq n_y}$ will be called the time sequence *of y.*

Let us note $dy(t)$ the derivative of y at time t. In what follows, any signal in the observed trace will be assumed to be f.p.l.c. Such a signal will be represented by its sequence $(t_i, y(t_i), dy(t_i))_{i<n_y}$, along with cut-off time t_{n_y}. As we assume continuity, this representation is slightly redundant, but facilitates the splitting of signals into segments. We may in addition require the limit of y at t_{n_y}, for which we abuse the notation and simply write $y(t_{n_y})$.

We will see that for every operator, the quantitative semantics preserves the f.p.l.c. property of signals, so that we can always assume such signals as inputs of the calculation in the inductive step. Note that the continuity is clearly preserved by the sup and inf operations. Also, no new derivative value is created in the process, so that from a computational standpoint the overhead of handling derivatives is compensated by the interpolation speedup.

Boolean operators Computing the robustness signal of $\neg\varphi$ from that of φ is trivial. One can simply note that if the sequence $(t_i, y(t_i), dy(t_i))_{i<n_y}$ represents $\rho(\varphi, w, .)$ then the sequence $(t_i, -y(t_i), -dy(t_i))_{i<n_y}$ represents $\rho(\neg\varphi, w, .)$. For conjunction, let us take y and y' the robustness signals of φ and ψ respectively, producing z the robustness signal of $\varphi \wedge \psi$. We build the sequence $(r_i)_{i\leq n_z}$ containing the sampling points of y and y' where they are both defined, and the points where y and y' punctually intersect. Note that there are less than $n_y + n_{y'}$ such intersections and $n_y + n_{y'}$ sampling points, so that we have $n_z \leq 4 \cdot \max\{n_y, n_{y'}\}$. Now, for all $i < n_z$ we let, using the lexicographic order

$$\begin{pmatrix} z(r_i) \\ dz(r_i) \end{pmatrix}^t = \min\left\{ \begin{pmatrix} y(r_i) \\ dy(r_i) \end{pmatrix}^t, \begin{pmatrix} y'(r_i) \\ dy'(r_i) \end{pmatrix}^t \right\}$$

The resulting sequence $(r_i, z(r_i), dz(r_i))_{i<n_z}$ adequately represents $\rho(\varphi\wedge\psi, w, .)$.

Untimed eventually $\Diamond$ Although not primitive in the syntax, this operator is easily computed and will be used as a subroutine for the until computation. We take y the robustness signal of φ, $(t_i)_{i<n_y}$ its time sequence, and z the robustness signal z of $\Diamond\,\varphi$. For any t in its definition domain $z(t) = \sup\limits_{t'\geq t} y(t')$, and we have immediately the following property: $\forall s < t,\ z(s) = \max\{\sup\limits_{[s,t)} y, z(t)\}$.
The step computation can be derived by applying the property at $t = t_{i+1}$ the time of some sample $i + 1 < n_y$. Depending on the possible orderings of $\{y(t_i), y(t_{i+1}), z(t_{i+1})\}$ there are four possibilities,

$$\begin{aligned}
y(t_i) \leq y(t_{i+1}) :&\quad \forall s \in [t_i, t_{i+1}),\ z(s) = \max\{y(t_{i+1}), z(t_{i+1})\} \\
y(t_i) > y(t_{i+1}) \geq z(t_{i+1}) :&\quad \forall s \in [t_i, t_{i+1}),\ z(s) = y(s) \\
z(t_{i+1}) \geq y(t_i) > y(t_{i+1}) :&\quad \forall s \in [t_i, t_{i+1}),\ z(s) = z(t_{i+1}) \\
y(t_i) > z(t_{i+1}) > y(t_{i+1}) :&\quad \exists t^*\, \forall s \in [t_i, t^*),\ z(s) = y(s) \text{ and} \\
&\qquad\quad \forall s \in [t^*, t_{i+1}),\ z(s) = z(t_{i+1})
\end{aligned}$$

The induction is initialized by substituting $\perp$ for $z(t_{n_y})$ in the property. Over the whole signal y, each sample t_i of y generates up to two samples in z, so that $n_z \leq 2 \cdot n_y$.

Until We treat the case of timed until by rewriting it into untimed until and timed eventually. This decomposition, due to [DT04] has been successfully applied to monitoring STL for the Boolean semantics. We have proved (Lemma 1) that it also holds for the quantitative semantics. For unbounded until $\mathbf{U}_{[a,+\infty)}$ we can use directly the second rewrite rule, whereas for bounded until we have $\varphi\,\mathbf{U}_{[a,b]}\,\psi \;\sim\; \Diamond_{[a,b]}\,\psi \wedge \Box_{[0,a]}\,(\varphi\,\mathbf{U}\,\psi)$ by first and second rewrite rules. *Globally* being the dual of *eventually*, it only remains to develop an algorithm for $\Diamond_{[a,b]}$ with $[a, b]$ a non-singular interval, along with an algorithm for the untimed until. These more involved computations are the object of the next section.

4 Algorithms

From a close examination of operators $\neg$, $\wedge$ and $\Diamond$ just achieved, we can immediately derive the corresponding Compute() algorithms with time-complexity linear in the number of samples of their input signals. By duality we also have an algorithm for $\vee$ with the same property. We now give detailed algorithms for the remaining two operators: $\mathbf{U}$, and $\Diamond_{[a,b]}$.

For any signal y and two time instants $s < t \in \mathbb{R}^+$ we note $y_{\restriction[s,t)}$ the restriction of y to the time interval $[s,t)$. The output signal z will be computed as a series of segments $z_{\restriction[s,t)}$ for s,t extracted from the time sequence of the input signals.

Operator U Let y and y' be the robustness signals of φ and ψ respectively, with $(t_i)_{i\leq n_y}$ and $(t'_i)_{i\leq n_{y'}}$ their respective time sequences. The calculation outputs z, the robustness signal of $\varphi\,\mathbf{U}\,\psi$ relative to w. By definition we have $z(t) = \sup\limits_{\tau\in[t,+\infty)} \min\{y'(\tau), \inf\limits_{[t,\tau]} y\}$.

Similarly to the Boolean semantics [MN04], the computation can be done by backward induction. Let $s < t$ be two times in $\mathrm{dom}(\varphi\,\mathbf{U}\,\psi)$. If we define $z_t(s) := \sup\limits_{\tau\in[s,t)} \min\{y'(\tau), \inf\limits_{[s,\tau]} y\}$, by the general properties of sup and inf we obtain the following inductive formula:

$$z(s) = \max\left\{z_t(s), \min\{\inf_{[s,t)} y, z(t)\}\right\}$$

Suppose that y is affine on the interval $[s,t]$.

- If $dy(s) \leq 0$ then $\forall\tau\in[s,t)$, $\inf\limits_{[s,\tau]} y = y(\tau)$. Thus $z_t(s) = \sup\limits_{\tau\in[s,t)} \min\{y'(\tau), y(\tau)\}$, and $z(s) = \max\left\{z_t(s), \min\{y(t), z(t)\}\right\}$
- Otherwise $\forall\tau\in[s,t)$, $\inf\limits_{[s,\tau]} y = y(s)$. Therefore $z_t(s) = \sup\limits_{\tau\in[s,t)} \min\{y'(\tau), y(s)\} = \min\{y(s), \sup\limits_{[s,t)} y'\}$, and $z(s) = \max\left\{z_t(s), \min\{y(s), z(t)\}\right\}$.

We let $t = t_i$ in the above, and compute $z(s)$ for all s on the segment $[t_i, t_{i+1})$. Taking the notation $\overline{v}$ for the constant signal of value v, we can now express all the operations involved as computations previously implemented. This gives us Algorithm 2, written under the simplifying assumption that $[t_0, t_{n_y}) \subseteq \mathrm{dom}(\psi, w)$, which can always be achieved by interpolation of y at $t'_0, t'_{n_{y'}}$ and sample renumbering if necessary.

Lemma 2. *The time-complexity of Algorithm 2 is linear in* $\max\{n_y, n_{y'}\}$.

Proof. The algorithm takes n_y steps. Step i computes the signal z on the interval $[t_i, t_{i+1})$ from partial signals $y_{\restriction[t_i,t_{i+1})}$ and $y'_{\restriction[t_i,t_{i+1})}$ along with two constant signals. Each step uses algorithms linear in the size of their inputs, so that the execution takes time linear in the sum of the size of the inputs of each step which is at most $3\cdot n_y + n_{y'}$. Thus the total execution time is linear in $\max\{n_y, n_{y'}\}$.

Algorithm 2 Compute($\mathbf{U}$, y, y')

```
$z_0 := \overline{\bot}$
$i := n_y - 1$
**while** $i \geq 0$ **do**
  **if** $dy(t_i) \leq 0$ **then**
    $z_1 :=$ Compute($\wedge, y'_{\restriction[t_i,t_{i+1})}, y_{\restriction[t_i,t_{i+1})}$)
    $z_2 :=$ Compute($\Diamond$ , $z_1$)
    $z_3 :=$ Compute($\wedge, y_{\restriction[t_i,t_{i+1})}, z_0$)
    $z_{\restriction[t_i,t_{i+1})} :=$ Compute($\vee, z_2, z_3$)
  **else**
    $z_1 :=$ Compute($\Diamond$ , $y'_{\restriction[t_i,t_{i+1})}$)
    $z_2 :=$ Compute($\wedge, \overline{z_1, y_{\restriction[t_i,t_{i+1})}}$)
    $z_3 :=$ Compute($\wedge, y(t_{i+1}), z_0$)
    $z_{\restriction[t_i,t_{i+1})} :=$ Compute($\vee, z_2, z_3$)
  **end if**
  $i := i - \underline{1}$
  $z_0 := \overline{z(t_{i+1})}$
**end while**
**return** $z$
```

Operator $\Diamond_{[a,b]}$ Let y be the robustness signal of φ with respect to w, with $(t_i)_{i \leq n_y}$ its time sequence. For a given $I = [a, b]$ we want to compute $z : t \mapsto \sup_{t+I} y$ the robustness signal of $\Diamond_I \varphi$ with respect to w.

Let t be a given time instant in dom$(\Diamond_I \varphi, w)$. Due to the f.p.l.c. hypothesis on y, one can easily see that there exist $t^* \in t+I$ such that $y(t^*) = z(t)$. Moreover it is sufficient to consider candidates for the maximum in the time sequence of y, along with $t + a$ and $t + b$. Namely

$$\sup_{t+[a,b]} y = \max \{y(t + a), y(t + b)\} \cup \{y(t_i) \mid t_i \in t + (a, b]\}$$

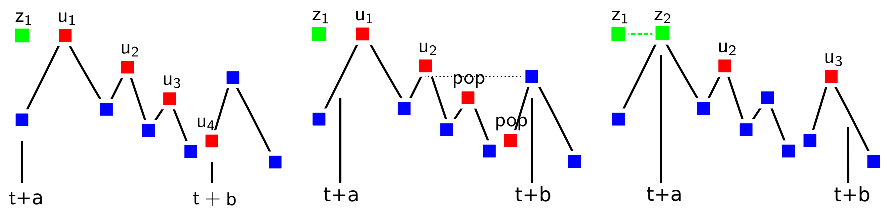
The problem of computing z is thus reduced to computing the maximum of $\{y(t_i) \mid t_i \in t + (a, b]\}$ when non empty, followed by a pointwise maximum with $y(t+a)$, $y(t+b)$. Intuitively, time intervals where $\{y(t_i) \mid t_i \in t+(a, b]\}$ provide the maximum corresponding to "plateau" phases, where the supremum is reached at a point in the interior of the interval $t + I$. On the other hand intervals where $y(t + a)$ or $y(t + b)$ give the maximum correspond to descending and ascending phases, respectively.

The maximum of $\{y(t_i) \mid t_i \in t+(a, b]\}$ can be computed by a straightforward adaptation of the running maximum filter algorithm given by [Lem06]. This work addresses the problem of computing, for signals over time domain $\mathbb{N}$, the maximum over a shifting window consisting of k elements. It it the first algorithm with time complexity linear in the length of the signal and *independent* of the window size k. We generalize this algorithm to the case of variable time-step.

The main idea is to maintain, as we increase t, a set of indices M, so-called a monotonic edge, such that

$$i \in M \text{ iff } t_i \in t + I \text{ and for all } t_j > t_i \text{ in } t + I,\ y(t_j) < y(t_i)$$

In particular for any given t, if $M \neq \emptyset$ we have $y(t_{\min M}) = \max\{y(t_i) \mid t_i \in t + (a, b]\}$. Assume M is known for a given time s, and is non empty. We begin by finding the first $t > s$ so that a either a new point appears at $t + b$ or some maximum candidate in M disappears at $t+a$, or both. We update M accordingly: if $t_{\min M} = t + a$ we remove $\min M$, the first index from M. Then if $t + b = t_i$ for a given i then we compare $y(t_i)$ with $y(t_k)$ for $k \in M$ in decreasing order of k, starting with the last candidate $y(t_{\max M})$. If $k \in M$ is so that $t_k \leq t_i$ then t_k is removed from M as outperformed by t_i, otherwise we stop. At this point i is inserted as the new last element of M. We now have in M an ordered set of maximum candidates in $t+I$; we can output $y(t_{\min M})$ and repeat the procedure for the next event. The algorithm steps are illustrated in Figure 1.



Step 1 Step 2 Step 3

Fig. 1. Steps of the Lemire algorithm. Initially, M contains indexes u_1, u_2, u_3, u_4. At step 2, a new value appears at $t + b$ which removes u_3 and u_4. At step 3, $t + a$ reaches u_1 which is removed from M.

We obtain the full algorithm, for piecewise linear signals, by integrating this value with $y(t+a)$ and $y(t+b)$. Let us note $y_a : t \mapsto y(t+a)$ and $y_b : t \mapsto y(t+b)$; such signals can be computed by simple shift of the time sequence of y. We see $y(t_{\min M})$ as a constant signal, noted $\overline{y(t_{\min M})}$. We then use the $\vee$ algorithm to take the pointwise maximum of y_a, y_b and $\overline{y(t_{\min M})}$. Note that on a step interval $[s, t)$ of our computation, these three signals are always affine. The pseudo-code of Algorithm 3 details the overall operation. For simplicity's sake we ignore the case whereby $M = \emptyset$ (occurs if $[a, b]$ is finer than some time step). In such a case we would only need to place the next t_i in M and output the pointwise maximum of y_a, y_b on the current segment.

Lemma 3. *The time-complexity of Algorithm 3 is linear in n_y.*

Proof. We begin by noticing that the computation of signals y_a, y_b and y' use previous algorithms linear in n_y. For each step, the integration of y' with the

Algorithm 3 Compute($\Diamond_{[a,b]}$, y)

```
$y_a := \mathrm{Shift}(y, -a)$, $y_b := \mathrm{Shift}(y, -b)$, $y' := \mathrm{Compute}(\vee, y_a, y_b)$
$s := t_0 - b,\ t := s,\ i := 0,\ M := \{0\}$
while $t + a < t_{n_y}$ do
  $t := \min\{t_{\min M} - a, t_{i+1} - b\}$
  if $t = t_{\min M} - a$ then
    $M := M \setminus \{\min M\}$
    $s := t$
  end if
  if $t = t_{i+1} - b$ then
    while $y(t_{i+1} - b) \geq y(t_{\max M})$ and $M \neq \emptyset$ do
      $M := M \setminus \{\max M\}$
    end while
    $M := M \cup \{i + 1\}$
    $i := i + 1$
  end if
  if $s \geq t_0$ then
    $z_{\restriction [s,t)} := \mathrm{Compute}(\vee, y'_{\restriction [s,t)}, \overline{y(t_{\min M})})$
  end if
end while
return $z$
```

constant signal $\overline{y(t_{\min M})}$ over the whole domain involves at most 3 samples; there are at most $2 \cdot n_y$ steps, so that the overall cost of these operations is linear in n_y. All that is left to show is that maintaining M throughout the computation takes time linear in n_y.

Storing M as a doubly-linked queue, we keep a sorted array and the elements to be accessed are always at the front or the back. Therefore we can consider the cost of each operation on M as unitary, and independent of the size of M. Under this hypothesis the cost of computing M is proportional to the number of value comparisons involved.With the same argument as [Lem06] we notice that on the whole run, there are n_y elements entering M and thus there are also n_y elements are leaving M. Each time the comparison $y(t_{i+1} - b) \geq y(t_{\max M})$ evaluates to true, an element is removed from M so there can only be n_y such comparisons that evaluate to true. When it evaluates to false we leave the loop hence there are at most n_y such comparisons that evaluate to false. Over the whole execution Algorithm 3 uses at most $2 \cdot n_y$ such value comparisons, making its execution time linear in n_y.

5 Complexity

To keep the discussion short, we restrict the syntax to primitive connectors $\{\mathrm{true}, x_i \geq 0, \neg, \wedge, \mathbf{U}_I\}$. The complexity results of previous sections can be summed up by stating the following: there exists a constant A such that for any signals y, y' and any connector $\neg, \wedge$ or $\mathbf{U}_I$, the corresponding Compute() algorithm takes execution time smaller than $A \cdot \max\{n_y, n_{y'}\}$.

We are interested in the time complexity of the robustness computation with respect to both trace and formula size. A trace $w = \{x_1, x_2, ..., x_k\}$ will have for size $|w| := \max\{n_{x_1}, n_{x_2}, ...n_{x_k}\}$, the maximum number of samples of its signals. A formula φ is represented by its syntactical tree, in which each node is an STL operator. A path in the syntactical tree has a *length* taken to be the number of binary connectors $\wedge$ and $\mathbf{U}_I$ it contains. The height of the formula $\mathrm{h}(\varphi)$ is then defined as maximum length for paths in the tree of φ. Note that our definition of height ignores atoms and negations. The size of a formula $|\varphi|$ will be simply defined as the number of nodes in the tree of φ.

Theorem 3. *There exists a constant d such that for any φ, w, the signal $z := \rho(\varphi, w, .)$ has a number of samples $n_z \leq d^{\mathrm{h}(\varphi)} \cdot |w|$*

Proof. If we take y, y' two signals and z be one of $t \mapsto \min\{y(t), y'(t)\}$ or $t \mapsto \sup_{\tau \in t+[a,b]} \min\{y'(\tau), \inf_{[t,\tau]} y\}$, then by immediate consequence of the existence of linear time algorithms to compute such signals there exists d such that $n_z \leq d \cdot \max\{n_y, n_{y'}\}$. We now prove the property by induction on the structure of φ.

For atomic formulas, the height is 0 while the robustness signal has a number of sampling points at most the size of the input trace. For the negation, we have $h(\neg\varphi) = h(\varphi)$ by definition while the number of samples is unchanged, we conclude by the induction hypothesis. To finish we examine the induction step for conjunction, the case of timed until being identical.
By definition we have $\mathrm{h}(\varphi \wedge \psi) = \max\{\mathrm{h}(\varphi), \mathrm{h}(\psi)\} + 1$. Let y, y' and z be the robustness signals of φ, ψ and $\varphi \wedge \psi$ with respect to w. By induction hypothesis we have $n_y \leq d^{\mathrm{h}(\varphi)} \cdot |w|$, and $n_{y'} \leq d^{\mathrm{h}(\psi)} \cdot |w|$. From the introductory remark $n_z \leq d \cdot \max\{d^{\mathrm{h}(\varphi)} \cdot |w|, d^{\mathrm{h}(\psi)} \cdot |w|\} = d \cdot d^{\max\{\mathrm{h}(\varphi),\mathrm{h}(\psi)\}} \cdot |w| = d^{\mathrm{h}(\varphi\wedge\psi)} \cdot |w|$.

Corollary 1. *The algorithm Robustness(φ, w) has time-complexity in $\mathcal{O}(|\varphi| \cdot d^{\mathrm{h}(\varphi)} \cdot |w|)$.*

Proof. Let φ be an arbitrary formula, and w an arbitrary trace. By Theorem 3, each subformula ψ of φ has a robustness signal with at most $d^{\mathrm{h}(\psi)} \cdot |w|$ sampling points, which is smaller or equal to $d^{\mathrm{h}(\varphi)} \cdot |w|$ in particular. Thus for each node of the tree of φ the corresponding Compute() algorithm takes execution time at most $A \cdot d^{\mathrm{h}(\varphi)} \cdot |w|$. There are exactly $|\varphi|$ such nodes, so that the main robustness computation of φ with respect to w is achieved in time at most $A \cdot |\varphi| \cdot d^{\mathrm{h}(\varphi)} \cdot |w|$.

The next example will convince the reader that the robustness signal size can indeed increase exponentially with the height of the formula.

Example 1. Let w_0 be the trace with signal x, defined over $[0, 8)$ and represented by the sequence $\big(t_i, x(t_i), dx(t_i)\big)_{i<4} := \{(0, 0, 0), (5, 0, -1), (6, -1, 0), (7, -1, 1)\}$. We define by induction the formula sequence $(\varphi_k)_{k\in\mathbb{N}}$ by

$$\varphi_0 := x \geq 0 \qquad \varphi_{k+1} := \Box_{[0,\frac{1}{2^{k+1}}]} (\Diamond_{[0,\frac{1}{2^{k-1}}]} \varphi_k \wedge \Diamond_{[\frac{1}{2^{k-1}},\frac{1}{2^{k-2}}]} \varphi_k)$$

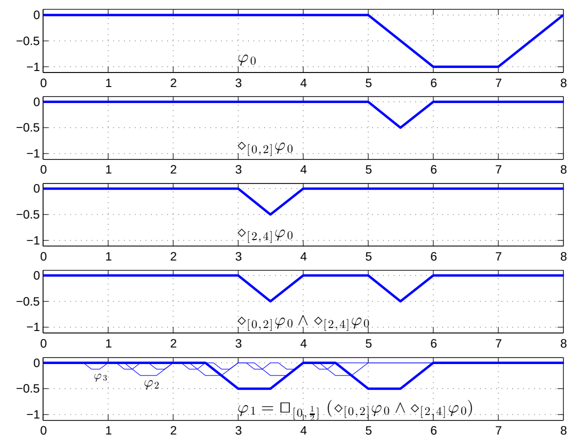
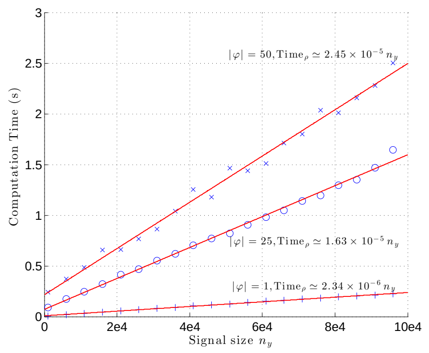


Fig. 2. Some steps in the computation of $\rho(\varphi_k, w_0, .)$

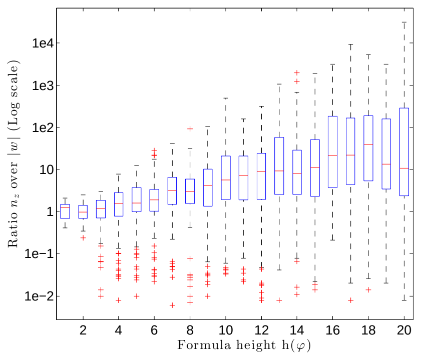
One can easily show (see Figure 2) that n_k the number of samples of $\rho(\varphi_k, w_0, .)$ is $2^k \cdot |w_0|$, while $\mathrm{h}(\varphi_k) = 3 \cdot k$, so that we have $n_k = (\sqrt[3]{2})^{\mathrm{h}(\varphi_k)} \cdot |w_0|$. Note that this example entails the same behaviour for the size of the *satisfaction* signals with respect to the formula height, so that traditional Boolean monitoring also appears to suffer some level of exponential complexity to this respect.

6 Experiments

The proposed algorithm has been implemented in C++ and interfaced with the STL parser of Breach. To plot Figure 3-(a), we computed the robustness estimate for three formulas of size 1, 25 and 50 for random signals of size ranging from 10^4 to 10^5 samples. We could confirm that for each formula, the computation time is linear with respect to the signal size. For Figure 3-(b), we fixed $n_y = 1000$ and generated 20 times 100 formulas of height $\mathrm{h}(\varphi)$ ranging from 1 to 20 and as many signals of size n_y. Our goal was to explore experimentally the result of Theorem 3. For each pair of formula and signal, we computed the robust signal and the ratio $\alpha = \frac{n_z}{ny}$ between its samples size n_z and the input signal size $|w| = n_y$. We first observe that for each value of $h(\varphi)$, there is high variability of this ratio. For instance, for $\mathrm{h}(\varphi) = 20$, α can vary from 10^{-2} to more than 10^4. Secondly, we observe that as predicted by Theorem 3, α appears to be bounded by some exponential $d^{\mathrm{h}(\varphi)}$, where we can roughly estimate $d \simeq 1.7$. Finally, the average value of α on our experiment seems also to follow an exponential function of $\mathrm{h}(\alpha)$, though with a lower constant $d \simeq 1.12$. This seems to indicate that exponential complexity is rather the rule than the exception. However, the conditions of these experiments (random signals, formula maximizing heights) are likely to be much more chaotic than for real systems and specifications. In particular, other experiments (not reported here) suggest that "high" formulas resulting mostly from large numbers of conjunctions do not exhibit an exponential behavior, so that the complexity mostly arise from nested temporal operators. And we believe that human-written specifications are not likely to contain deeply nested temporal operators, as the intuition of such formula is generally hard to grasp.



(a) Compution time against signal size



(b) Size increase of robust signal against $\|\varphi\|$.

Fig. 3. Experimental validation of the complexity results given in Section 5.

Next we compared the performance of our algorithm with that of the DP-TaLiRo [FSUY12], based on a dynamic programming approach and implemented in S-TaLiRo version 1.3. We compared the monitoring of $\Diamond_I$ and $\mathbf{U}_I$ for signals of various sizes and different time intervals I.[3] The results are given in Table 1. Except for signals of small size (100-1000), Breach is consistently faster by several orders of magnitude. One partial explanation could be the fact that in the framework of TaLiRo, the robust satisfaction of a predicate is obtained through the computation of a distance function, which is done by the monitoring algorithm, leading to an additional "hidden cost" [FSUY12]), whereas in our case, this computation is separated from the monitoring.[4] However, all the predicates in our experiments are such that the distance function should be trivial to compute so this alone cannot account for the difference in the performance. Also, our results confirmed that the computation time for bounded time operators does not depend on the size of the time interval, as in [Lem06], which is an improvement over the complexity of the DP-TaLiRo algorithm.

[3] Note that a comparison of nested formulas would make less sense since DP-TaLiRo is implemented for fixed time-steps, thus the number of samples cannot vary from one level to another.

[4] In Breach, predicates are of the general form $f(x) > 0$ and here we considered that f was computed beforehand to produce the signals $y(t) = f(x(t))$ as inputs to our

Computation times (s) of robustness estimates for $\Diamond_I$ and $\mathbf{U}_I$								
	DP-Taliro				Breach			
n_y	$1e2$	$1e3$	$1e4$	$1e5$	$1e2$	$1e3$	$1e4$	$1e5$
$\Diamond_{[1,2]}$	0.001091	0.00278	0.176	16.6	0.00312	0.00302	0.004	0.0193
$\Diamond_{[1,11]}$	0.000689	0.00304	0.212	20.4	0.00286	0.00262	0.00391	0.0173
$\Diamond_{[1,21]}$	0.000713	0.00334	0.253	24.3	0.00268	0.00269	0.00412	0.0185
$\Diamond_{[1,31]}$	0.000707	0.0038	0.278	27.3	0.00302	0.00281	0.00409	0.0208
$\mathbf{U}_{[1,2]}$	0.523	4.72	46.8	486	0.00577	0.00766	0.0268	0.228
$\mathbf{U}_{[1,11]}$	0.482	4.55	47.1	493	0.00567	0.00743	0.0269	0.223
$\mathbf{U}_{[1,21]}$	0.468	4.59	46.2	499	0.00545	0.00722	0.0268	0.229
$\mathbf{U}_{[1,31]}$	0.462	4.7	46.7	505	0.00567	0.0073	0.0274	0.222

Table 1. Experimental comparison with DP-Taliro algorithm.

7 Discussion and Future Work

We developed a new algorithm computing robust satisfaction of STL formulas by piecewise-linear signals. The algorithm is linear in the size of the signal as measured by the number of sampling points. The algorithm extends to dense time the algorithm of [Lem06] for maintaining the maximal value of a numerical sequence over a shifting window and its implementation confirms its theoretical properties. Our implementation could handle signals with millions of samples and formulas with tens of operators. It remains to be seen whether the worst-case exponential growth in the size of the formula occurs for real-life formulas rather than the random formulas we experimented with. Another important direction to investigate would be the design of an online variant of our algorithm, which is by nature offline.

References

ALFS11. Y. Annpureddy, C. Liu, G.E. Fainekos, and S. Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *TACAS*, 2011.

CGP99. E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, 1999.

DFG$^+$11. A. Donzé, E. Fanchon, L. M. Gattepaille, O. Maler, and P. Tracqui. Robustness analysis and behavior discrimination in enzymatic reaction networks. *PLoS ONE*, 6(9), 2011.

DM10. A. Donzé and O. Maler. Robust satisfaction of temporal logic over real-valued signals. In *FORMATS*, pages 92–106, 2010.

DMB$^+$12. A. Donzé, O. Maler, E. Bartocci, D. Nickovic, R. Grosu, and S. Smolka. On temporal logic and signal processing. In *ATVA*, 2012.

algorithm. Note that this is slightly more general than TaLiRo as f can implement a distance function [JDDS13]).

Don10. A. Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *CAV*, pages 167–170, 2010.

DT04. D. D'Souza and N. Tabareau. On timed automata with input-determined guards. In *FORMATS/FTRTFT*, 2004.

FP09. G.E. Fainekos and G.J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42), 2009.

FSUY12. G.E. Fainekos, S. Sankaranarayanan, K. Ueda, and H. Yazarel. Verification of automotive control applications using s-taliro. In *ACC*, 2012.

JDDS13. Xiaoqing Jin, Alexandre Donzé, Jyotirmoy Deshmukh, and Sanjit Seshia. Mining requirements from closed-loop control models. In *HSCC'13*, 2013.

JKN10. K.D. Jones, V. Konrad, and D. Nickovic. Analog property checkers: a DDR2 case study. *Formal Methods in System Design*, 36(2), 2010.

Lem06. D. Lemire. Streaming maximum-minimum filter using no more than three comparisons per element. *CoRR*, abs/cs/0610046, 2006.

MDMF12. N. Mobilia, A. Donzé, J.-M. Moulis, and E. Fanchon. A model of the cellular iron homeostasis network using semi-formal methods for parameter space exploration. In *HSB*, 2012.

MN04. O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *FORMATS/FTRTFT*, pages 152–166, 2004.

MN12. O. Maler and D. Nickovic. Monitoring properties of analog and mixed-signal circuits. *Software Tools for Technology Transfer*, 2012.

MNP08. O. Maler, D. Nickovic, and A. Pnueli. Checking temporal properties of discrete, timed and continuous behaviors. In *Pillars of Computer Science*, 2008.

MP91. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag New York, 1991.

MP95. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag New York, 1995.

Nil86. N.J. Nilsson. Probabilistic logic. *Artificial intelligence*, 28(1):71–87, 1986.

NM07. D. Nickovic and O. Maler. AMT: A property-based monitoring tool for analog systems. In *FORMATS*, pages 304–319, 2007.

Pnu77. A. Pnueli. The temporal logic of programs. In *Proc. 18th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 46–57, 1977.

QS82. J. P. Queille and J. Sifakis. Specification and Verification of Concurrent Systems in CESAR. In *5th Int. Symp. on Programming*, 1982.

RBFS08. A. Rizk, G. Batt, F. Fages, and S. Soliman. On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology. In *CMSB*, 2008.

Zad65. L. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.