Quantitative Monitoring of STL with Edit Distance

Stefan Jakšić[1,2(✉)], Ezio Bartocci[2], Radu Grosu[2], and Dejan Ničković[1]

[1] AIT Austrian Institute of Technology, Seibersdorf, Austria
Stefan.Jaksic.fl@ait.ac.at
[2] Faculty of Informatics, Vienna University of Technology, Vienna, Austria

Abstract. In cyber-physical systems (CPS), physical behaviors are typically controlled by digital hardware. As a consequence, continuous behaviors are discretized by sampling and quantization prior to their processing. Quantifying the similarity between CPS behaviors and their specification is an important ingredient in evaluating correctness and quality of such systems. We propose a novel procedure for measuring robustness between digitized CPS signals and Signal Temporal Logic (STL) specifications. We first equip STL with quantitative semantics based on the *weighted edit distance* (WED), a metric that quantifies both space and time mismatches between digitized CPS behaviors. We then develop a dynamic programming algorithm for computing the robustness degree between digitized signals and STL specifications. We implemented our approach and evaluated it on an automotive case study.

1 Introduction

Cyber-physical systems (CPS) integrate heterogeneous collaborative components that are interconnected between themselves and their physical environment. They exhibit complex behaviors that often combine discrete and continuous dynamics. The sophistication, complexity and heterogeneity of CPS makes their verification a difficult task. Runtime monitoring addresses this problem by providing a formal, yet scalable, verification method. It achieves both rigor and efficiency by enabling evaluation of systems according to the properties of their individual behaviors.

In the recent past, property-based runtime monitoring of CPS centered around Signal Temporal Logic (STL) [18] and its variants have received considerable attention [1,5,6,10–12,20]. STL is a formal specification language for describing properties of continuous and hybrid behaviors. In its original form, STL allows to distinguish correct from incorrect behaviors. However, the binary true/false classification may not be sufficient for real-valued behaviors. In fact, systems with continuous dynamics are often sensitive to small perturbations in initial conditions, system parameters and the accuracy of sensors, which may influence the correctness of the verdict. In order to address this problem, the satisfaction relation can be replaced by the *robustness degree* [10–12] of a behavior with respect to a temporal specification. The robustness degree gives a finer measure of how far is the behavior from satisfying or violating of the specification.

Y. Falcone and C. Sanchez (Eds.): RV 2016, LNCS 10012, pp. 201–218, 2016.
DOI: 10.1007/978-3-319-46982-9_13

In this paper, we propose a novel quantitative semantics for STL that measures the behavior mismatches in both *space* and *time*. We consider applications in which continuous CPS behaviors are observed by a digital device. In this scenario, continuous behaviors are typically discretized, both in time and space, by an analog-to-digital converter (ADC). As a consequence, we interpret STL over discrete-time digitized behaviors. We first define the *weighted* edit distance as an appropriate metric for measuring combined space-time similarity between discretized CPS behaviors. We then provide the quantitative semantics for STL based on this distance and discuss the effects of sampling and quantization on the distance value. We develop an efficient on-line algorithm for computing the robustness degree between a behavior and a STL formula. The algorithm can be directly implemented both in software and hardware. In the former case, the implemented procedure can be connected to the simulation engine of the CPS design and used to monitor its correctness and quality. In the latter case, the resulting implementation can be deployed on the Field Programmable Gate Array (FPGA) and used to monitor real systems or design emulations. We implement the above procedure in Verilog and evaluate it on an automotive benchmark.

Related Work. The Levenshtein (edit) distance [17] has been extensively used in information theory, computer science and bioinformatics for many applications, including approximate string matching, spell checking and fuzzy string searching. Levenshtein automata [24] were introduced to reason about the edit distance from a reference string. A Levenshtein automaton of degree n for a string w recognizes the set of all words whose edit distance from w is at most n. A dynamic programming procedure for computing the edit distance between a string and a regular language has been proposed in [26]. The problem of computing the smallest edit distance between any pair of distinct strings in a regular language has been studied in [15]. In contrast to our work, these classical approaches to edit distance consider only operations with simple weights on unordered alphabets and do not relate the distance to specification formalisms.

The edit distance for weighted automata was studied in [19], where the authors propose a procedure for computing the edit distance between weighted transducers. A space efficient algorithm for computing the edit distance between a string and a weighted automaton over a tropical semiring was developed in [2]. The resulting approach is generic and allows for instance to assign an arbitrary cost to each substitution pair. However, all substitution pairs must be enumerated by separate transitions. In contrast, we consider signals with naturally ordered alphabets as input strings and hence can efficiently handle substitution over large alphabets by treating allowed input values with symbolic constraints. In addition, we use the edit distance to define the semantics of a temporal specification formalism. Finally, we provide insights into the effect of sampling and quantization to the computation of the distance. The weighted Hamming and edit distances between behaviors are also proposed in [23], where the authors use it to develop procedures for reasoning about the introduce the Lipshitz-robustness of Mealy machines and string transducers. The notion of robustness

is different from ours, and in contrast to our work it is not computed against a specification.

The quantitative semantics for temporal logics were first proposed in [12,22], with the focus on the *spatial* similarity of behaviors, given by their point-wise comparison. The spatial quantitative semantics is sensitive to phase shifts and temporal inaccuracies in behaviors – a small temporal shift in the behavior may result in a large robustness degree change. This problem was addressed in [11], in which STL with spatial quantitative semantics is extended with time robustness. In [1], the authors propose another approach of combining space and time robustness, by extending STL with *averaged* temporal operators. Another approach to determining robustness of hybrid systems using self-validated arithmetics is shown in [13]. Monitoring of different quantitative semantics is implemented in tools S-TaLiRo [3] and Breach [9]. These works differ from ours in that they all assume continuous behaviors, in contrast to our approach where behaviors are quantized and sampled.

The recent results on using Skorokhod metric to compute the distance between piecewise-linear or piecewise-constant continuous behaviors [8] partially inspired our work. Skorokhod metric quantifies both space and time mismatches between continuous behaviors by allowing application of time distortions in behaviors in order to minimize their pointwise distance. The distortion of the timeline is achieved by applying a retiming function - a continuous bijective strictly increasing function from time domain to time domain. Given a behavior $x(t)$, the resulting retimed behavior $r(x(t))$ preserves the values and their order but not the duration between two values. This information-preserving distance relies on continuous time and is not applicable to the discrete time domain – stretching and compressing the discrete time axis results inevitably in an information loss. Finally, computation of the Skorokhod distance was extended to the flow-pipes in [7], but we are not aware of any work that addrsses the problem of computing the Skorokhod distance between a behavior and a temporal specification.

2 Preliminaries

In this section, we provide the necessary definitions to develop the algorithm presented in subsequent sections of the paper. We first shortly recall the notion of metric spaces and distances. We then define signals and Signal Temporal Logic. Finally, we introduce a variant of symbolic and weighted symbolic automata.

Metric Spaces and Distances. A metric space is a set for which distances between all elements in the set are defined.

Definition 1 (Metric space and distance). *Suppose that $\mathcal{M}$ is a set and $d : \mathcal{M} \times \mathcal{M} \to \mathbb{R}$ is a function that maps pairs of elements in $\mathcal{M}$ into the real numbers. Then $\mathcal{M}$ is a* metric space *with the* distance measure *d, if (1) $d(m_1, m_2) \geq 0$ for all m_1, m_2 in $\mathcal{M}$; (2) $d(m_1, m_2) = 0$ if and only if $m_1 = m_2$; (3) $d(m_1, m_2) =$*

$d(m_2, m_1)$ *for all* m_1, m_2 *in* $\mathcal{M}$*; and (4)* $d(m_1, m_2) \leq d(m_1, m) + d(m, m_2)$ *for all* m, m_1, m_2 *in* $\mathcal{M}$*.*

Given $m \in \mathcal{M}$ and $M \subseteq \mathcal{M}$, we can lift the above definition to reason about the distance between an element m of $\mathcal{M}$ and the subset M of $\mathcal{M}$ as follows

$$d(m, M) = \min_{m' \in M} d(m, m')$$

We define the *robustness degree* $\rho(m, M)$ of m with respect to the set M as follows

$$\rho(m, M) = \begin{cases} d(m, \mathcal{M} \backslash M) \text{ if } m \in M \\ -d(m, M) \quad \text{otherwise} \end{cases}$$

Signals. Let X be a finite set of variables defined over some domain $\mathbb{D}$. Then, a *signal* s is a function $s : \mathbb{T} \times X \to \mathbb{D}$, where $\mathbb{T}$ is the time domain[1]. We distinguish between *analog*, *discrete* and *digital* signals. Analog signals have continuous value and time domains. The time domain of discrete signals is the set of integers, while digital signals have in addition their value domain restricted to a finite set. Digital signals can be obtained by *sampling* and *quantization* of analog signals. The conversion of analog to digital signals is at the core of the signal processing field and is in practice done by an *analog-to-digital converter* (ADC).

Sampling is the process of reducing the continuous time in analog signals to the discrete time in the resulting discrete signal. The ideal theoretical sampling function periodically measures the value of the analog signal every T time units, where T denotes the *sampling interval*. Similarly, we denote by f the *sampling frequency*, that is the average number of measurements obtained by sampling in one second, where $f = 1/T$. Given an analog signal $s_a : \mathbb{R}_{\geq 0} \times X \to \mathbb{R}^n$ and a sampling interval T, applying the ideal sampling function to s_a results in a discrete signal $s_{disc} : \mathbb{N} \times X \to \mathbb{R}$ such that $s_{disc}(i, x) = s_a(iT, x)$ for all $i \geq 0$ and $x \in X$.

When sampling real-valued signals, it is impossible to maintain the arbitrary precision of its values, which consequently must be restricted to a finite set. Quantization consists of converting real values to their discrete numerical approximations, and thus allows to map discrete to digital signals. We consider the basic uniform quantization function with a *quantization step* Q which is defined as follows

$$Q(r) = \mathsf{Q} \cdot \lfloor |r|/\mathsf{Q} + 0.5 \rfloor,$$

where $r \in \mathbb{R}$. We note that the quantization can be decomposed into two stages, *classification* and *reconstruction*. The classification function c maps the real input value into an integer index k, and the reconstruction function y converts k into the actual discrete approximation of the input. Hence, we have that $Q(r) = y(c(r))$ where

[1] We use $s(t)$ to denote the valuation vector of the variables in X at time t.

$$c(r) = \lfloor |r|/\mathsf{Q} + 0.5 \rfloor$$
$$y(k) = \mathsf{Q} \cdot k$$

The decomposition of the quantization into two independent stages has a practical advantage – without loss of generality, we can from now directly work with digital signals obtained after the classification stage with their value domain being a finite subset of $\mathbb{N}$. We also restrict ourselves to signals that have *finite-length* and hence are of the form $s_{dig} : [0, l) \times X \to [v_{min}, v_{max}]$, where $[0, l)$ and $[v_{min}, v_{max}]$ are intervals in $\mathbb{N}$, and X is now the set of variables defined over the domain $[v_{min}, v_{max}]$. We extend the signal notation $s(i, X)$ to denote the vector $\mathbb{D}^{|X|}$ of all variable values in X at time i. From now on, we refer to digital signals of finite length simply as signals and denote them by s.

Signal Temporal Logic. In this paper, we study Signal Temporal Logic (STL) with both *past* and *future* operators interpreted over digital signals of final length[2].

Let X be a finite set of variables defined over a finite interval domain $\mathbb{D} = [v_{min}, v_{max}] \subseteq \mathbb{N}$. We assume that X is a metric space equiped with a distance d. The syntax of a STL formula φ over X is defined by the grammar

$$\varphi := x \sim u \mid \neg\, \varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1\, \mathcal{U}_I\, \varphi_2 \mid \varphi_1\, \mathcal{S}_I\, \varphi_2$$

where $x \in X$, $\sim \in \{<, \leq\}$, $u \in \mathbb{D}$, I is of the form $[a, b]$ or $[a, \infty)$ such that $a, b \in \mathbb{N}$ and $0 \leq a \leq b$. The other standard operators are derived as follows: $\mathsf{true} = p \vee \neg p$, $\mathsf{false} = \neg\mathsf{true}$, $\varphi_1 \wedge \varphi_2 = \neg(\neg\, \varphi_1 \vee \neg\, \varphi_2)$, $\Diamond_I\ \varphi = \mathsf{true}\mathcal{U}_I\, \varphi$, $\Box_I\ \varphi = \neg\ \Diamond_I\ \neg\, \varphi$, $\diamondsuit_I\, \varphi = \mathsf{true}\mathcal{S}_I\, \varphi$, $\boxminus_I\, \varphi = \neg\ \diamondsuit_I \neg\, \varphi$, $\bigcirc\, \varphi = \mathsf{false}\mathcal{U}_{[1,1]}\, \varphi$ and $\ominus\, \varphi = \mathsf{false}\mathcal{S}_{[1,1]}\, \varphi$.

The semantics of a STL formula with respect to a signal s of length l is described via the satisfiability relation $(s, i) \models \varphi$, indicating that the signal s satisfies φ at the time index i, according to the following definition where $\mathbb{T} = [0, l)$.

$$\begin{aligned}
(s,i) &\models x \sim u && \leftrightarrow s(i,x) \sim u \\
(s,i) &\models \neg\, \varphi && \leftrightarrow (s,i) \not\models \varphi \\
(s,i) &\models \varphi_1 \vee \varphi_2 && \leftrightarrow (s,i) \models \varphi_1 \text{ or } (s,i) \models \varphi_2 \\
(s,i) &\models \varphi_1\, \mathcal{U}_I\, \varphi_2 && \leftrightarrow \exists j \in (i+I) \cap \mathbb{T} : (s,j) \models \varphi_2 \text{ and } \forall i < k < j, (s,k) \models \varphi_1 \\
(s,i) &\models \varphi_1\, \mathcal{S}_I\, \varphi_2 && \leftrightarrow \exists j \in (i-I) \cap \mathbb{T} : (s,j) \models \varphi_2 \text{ and } \forall j < k < i, (s,k) \models \varphi_1
\end{aligned}$$

We note that we use the semantics for $_I$ and $_I$ that is strict in both arguments and that we allow punctual modalities due to the discrete time semantics. Given an STL formula φ, we denote by $L(\varphi)$ the *language* of φ, which is the set of all signals s such that $(s, 0) \models \varphi$.

[2] Although this segment of STL is expressively equivalent to LTL, use the STL name to highlight the explicit notions of real-time and quantitative values in the language.

Automata and Weighted Automata. In this section, we define a variant of *symbolic automata* [25] and also introduce its *weighted* extension. Similarly to the definition of STL, we consider $\mathbb{D} = [v_{min}, v_{max}]$ to be the finite interval of integers equipped with the distance d and let X to be a finite set of variables defined over $\mathbb{D}$. The variable valuation $v(x)$ is a function $v : X \to \mathbb{D}$, which we naturally extend to the valuation $v(X)$ of the set X. A variable *constraint* γ over X is defined by the grammar in negation normal form $\gamma := x \leq c \mid \neg(x \leq c) \mid \gamma_1 \vee \gamma_2 \mid \gamma_1 \wedge \gamma_2$, where $x \in X$ and $c \in \mathbb{D}$. We denote by $\Gamma(X)$ the set of all constraints definable over X. Given the valuation $v(X)$ and a constraint γ over X, we write $v(X) \models \gamma$ when $v(X)$ satisfies γ.

Definition 2 (Symbolic Automata). *We define a* symbolic automaton $\mathcal{A}$ *as the tuple* $\mathcal{A} = (\mathbb{D}, X, Q, I, F, \Delta)$*, where* $\mathbb{D}$ *is the finite* alphabet*,* X *is a finite set of variables over* $\mathbb{D}$*,* Q *is a finite set of* states*,* $I \subseteq Q$ *is the set of* initial states*,* $F \subseteq Q$ *is the set of* final states *and* $\Delta = \Delta_X \cup \Delta_\epsilon$ *is the* transition relation*, where* $\Delta_X \subseteq Q \times \Gamma(X) \times Q$ *and* $\Delta_\epsilon \subseteq Q \times \{\epsilon\} \times Q$ *are sets of transitions that consume an input letter and* silent *transitions.*

Given a $q \in Q$, let $\mathcal{E}(q)$ denote the set of states reachable from q by following ϵ-transitions in Δ only. Formally, we say that $p \in \mathcal{E}(q)$ iff there exists a sequence of states $q_1, \ldots, q_k$ such that $q = q_1$, $(q_i, \epsilon, q_{i+1}) \in \Delta$ for all $0 \leq i < k$, and $p = q_k$. Let $s : [0, l) \times X \to \mathbb{D}$ be a signal. We say that s is a *trace* of $\mathcal{A}$ if there exists a sequence of states $q_0, \ldots, q_l$ in Q such that $q_0 \in \mathcal{E}(q)$ for some $q \in I$, for all $0 \leq i < l$, there exists $(q_i, \gamma, q) \in \Delta$ for some γ such that $s(i, X) \models \gamma$ and $q_{i+1} \in \mathcal{E}(q)$ and $q_l \in F$. We denote by $L(\mathcal{A})$ the set of all traces of $\mathcal{A}$. A *path* π in $\mathcal{A}$ is a sequence $\pi = q_0 \cdot \delta_0 \cdot q_1 \cdots \delta_{n-1} \cdot q_n$ such that $q_0 \in I$ and for all $0 \leq i < n$, δ_i is either of the form (q_i, γ, q_{i+1}) or (q_i, ϵ, q_{i+1}). We say that π is *accepting* if $q_n \in F$. Given a trace $s : [0, l) \times X \to \mathbb{D}$ and a path $\pi = q_0 \cdot \delta_0 \cdot q_1 \cdot \delta_1 \cdots \delta_{n-1} \cdot q_n$, we say that s induces π in $\mathcal{A}$ if π is an accepting path in $\mathcal{A}$ and there exists a monotonic injective function $f : [0, l) \to [0, n]$ such that for all $0 \leq i < l$, $\delta_i = (q_i, \gamma, q_{i+1})$ such that $s(i, X) \models \gamma$ and $\delta_j = (q_j, \epsilon, q_{j+1})$ for all $j \in \bigcup_{0 \leq i < n} (f(i), f(i+1)) \cup [0, f(0)) \cup (f(l), n]$. We denote by $\Pi(\mathcal{A}, s) = \{\pi \mid s \text{ induces } \pi \text{ in } \mathcal{A}\}$ the set of all paths in $\mathcal{A}$ induced by s.

We now introduce *weighted* symbolic automata, by adding a weight function to the transitions of the symbolic automaton, relative to the consumed input letter.

Definition 3 (Weighted symbolic automata). *A* weighted symbolic automaton $\mathcal{W}$ *is the tuple* $\mathcal{W} = (\mathbb{D}, X, Q, I, F, \Delta, \lambda)$*, where* $\mathcal{A} = (\mathbb{D}, X, Q, I, F, \Delta)$ *is a symbolic automaton and* $\lambda : \Delta \times (\mathbb{D}^{|X|} \cup \{\epsilon\}) \to \mathbb{Q}^+$ *is the weight function.*

Let s be a signal of size l and $\pi = q_0 \cdot \delta_0 \cdots \delta_{n-1} \cdot q_n$ a path in $\mathcal{W}$ induced by s. The value of π in $\mathcal{W}$ subject to s, denoted by $v(s, \pi, \mathcal{W})$, is the sum of weights associated to the transitions in the path π and subject to the signal s. We define the *value* $v(s, \mathcal{W})$ of s as the minimum value from all the paths in $\mathcal{W}$ induced by s, i.e. $v(s, \mathcal{W}) = \min_{\pi \in \Pi(\mathcal{W}, s)} v(s, \pi, \mathcal{W})$.

3 Weighted Edit Distance

Measuring the similarity of sequences is important in many application areas, such as information theory, spell checking and bioinformatics. The *Hamming distance* d_H is the most basic and common string measure arising from the information theory. It measures the minimum number of *substitution* operations needed to match equal length sequences. The *edit distance* d_E extends the Hamming distance with two additional operations, *insertion* and *deletion* and is defined as the minimum accumulation of edit operation costs used to transform one sequence into the other.

Neither of these metrics provide satisfactory solution for comparing digitized signals. They are defined over unordered alphabets and associate fixed costs to different kinds of operations. In contrast, the value domain of digital signals admits a natural notion of a distance representing the difference between two signal valuations. In addition, the Hamming distance provides only pointwise comparisons between sequences and consequently does not account for potential timing discrepancies in the sampled signals. Two discrete signals that differ only in a constant time delay will typically have a large Hamming distance. The edit distance addresses this problem by allowing us to bridge the time shifts using insertion and deletion operations.

Inspired by [19,23], we propose the *weighted edit distance* as the measure for comparing the similarity of two discrete signals. It adopts the insertion and deletion operations from the edit distance and adapts the substitution operation to the ordered alphabets. Since we consider multi-dimensional signals, we extend the cost of the substitution operation to take into account different variable valuations.

Let X be a finite set of variables defined over some interval domain $\mathbb{D} = [v_{min}, v_{max}]$. Given two valuation vectors $a, b \in \mathbb{D}^{|X|}$ of X, we denote by $d_M(a,b)$ the *Manhattan distance* [16] between a and b, where $d_M(a,b) = \Sigma_{i=0}^{|X|-1}|a_i - b_i|$. Let $w_i, w_d \in \mathbb{Q}$ be weight constants for the insertion and deletion operations. We then define the *costs* of the substitution c_s, insertion c_i and deletion c_d operations as follows: (1) $c_s(a,b) = d_M(a,b)$; (2) $c_i = w_i$; (3) $c_d = w_d$. The definition of the WED adapts the classical edit distance recursive definition with the new costs.

Definition 4 (Weighted edit distance). *Let* $s_1 : [0,l) \times X \to \mathbb{D}$ *and* $s_2 : [0,l) \times X \to \mathbb{D}$ *be discrete-time signals. The* weighted edit distance $d_W(s_1, s_2)$ *equals to* $d_{l,l}(s_1, s_2)$:

$$
\begin{aligned}
d_{-1,-1}(s_1,s_2) &= 0 \\
d_{i,-1}(s_1,s_2) &= d_{i-1,-1}(s_1,s_2) + c_i \\
d_{-1,j}(s_1,s_2) &= d_{-1,j-1}(s_1,s_2) + c_d \\
d_{i,j}(s_1,s_2) &= min \begin{cases} d_{i-1,j-1}(s_1,s_2) + c_s(s_1(i,X), s_2(j,X)) \\ d_{i-1,j}(s_1,s_2) + c_i \\ d_{i,j-1}(s_1,s_2) + c_d \end{cases}
\end{aligned}
$$

Proposition 1. *The weighted edit distance is a distance.*

Remark. We chose the Manhattan distance for the substitution cost because it combines the absolute difference of several signal components.

We first note that the distance is additive in two dimensions - the Manhattan distance adds substitution costs for each variable in the signal, and the edit operation costs are accumulated over the signal length. In addition, the distance is sensitive to the sampling period used to discretize the signal. As a consequence, this distance can be *normalized*, in order to provide more uniform results. Given signals s_1, s_2 of length l defined over X and sampled with a period T, the value domain $\mathbb{D} = [v_{max}, v_{min}]$, we define the normalized weigted edit distance, which is always bounded by $[0, 1]$ as follows:

$$d_W^{\#}(s_1, s_2) = \frac{T \cdot d_W(s_1, s_2)}{l|X|(v_{max} - v_{min})}.$$

3.1 Sampling, Quantization and Weighted Edit Distance

We compute the WED between digital signals resulting from physical behavior observations after sampling and quantization. In this section, we discuss the effect of inaccuracies introduced by these operations on the WED.

Let s be an analog signal, T a sampling period and Q a quantization step. We assume that s has a band limit f_M and $T \leq 1/(2f_M)$. We denote by $s[T]$ the discrete signal obtained from s by sampling with the period T, and by $s[T][\mathsf{Q}]$ the digital signal obtained from $s[T]$ by quantization with the step Q.

We cannot directly relate the WED to the analog signals, because it is not defined in continuous time. However, this distance allows tackling phase shifts in the sampled signals. Consider two analog signals $s_1(t)$ and $s_2(t-\tau)$ such that $\tau = iT$ for some $i \geq 0$ and their sampled variants $s_1[T](t)$ and $s_2[T](t)$. It is clear that with $2 \cdot i$ insertion and deletion operations, $s_2[T]$ can be transformed into $s_1[T]$ such that their remaining substitution cost equals to 0. This situation is illustrated in Fig. 1 (see signals s_1 and s_2). We see that the distance between the two signals initially grows due to the insertion and deletion operations, but that eventually it becomes perfectly stable.

Now consider another signal $s_3(t) = s_1(t - \tau)$ such that τ is not a multiple of T. In this case, the sampled signal $s_3[T](t)$ cannot be perfectly transformed into $s_1[T](t)$ by using insertion and deletion operations because of the mismatch between the sampling period and the phase shift. As a consequence, the distance between $s_1[T](t)$ and $s_3[T](t)$ will accumulate substitution costs due to this mismatch. This scenario is also depicted in Fig. 1 (see signals s_1 and s_3). The figure shows that after an initial steep increase of the distance due to the insertion and deletion operations, its value does not converge, but continues slowly increasing due to the accumulation of remaining substitution costs.

We define the *sampling error* as the maximum difference in value between two periods in a sampled signal. Intuitively, this value gives an impression about the error that can be accumulated when comparing sampled variants of two

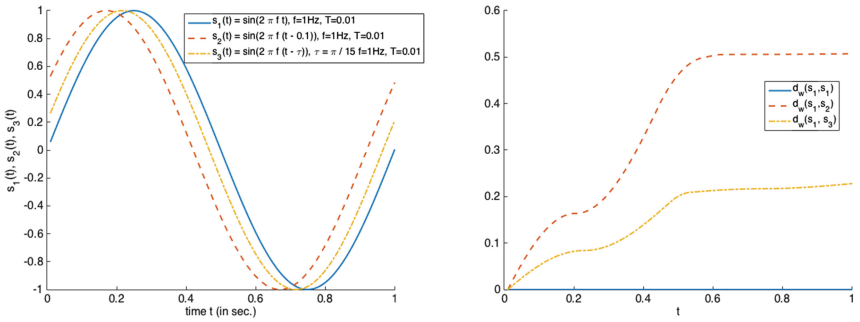

Fig. 1. Weighted edit distances $d_W(s_1, s_2)$ and $d_W(s_1, s_3)$, where $s_1(t) = \sin(2\pi ft)$, $s_2(t) = \sin(2\pi f(t - 0.1))$, $s_3(t) = \sin(2\pi f(t - \tau))$, $T = 0.01$, $f = 1\,Hz$ and $\tau = \pi/15$.

phase-shifted signals, when the phase shift is not a multiple of the sampling period.

Definition 5 (Sampling error). *Let s be an analog signal with band-limit f_M and sampling period T, such that $T \leq 1/(2f_M)$. The sampling error is defined as* $\mathit{err}_T(s, i) = ||s[T](i) - s[T](i + 1)| - \max_{0 \leq \tau \leq T} |s(iT) - s(iT + \tau)||$

We show that this error converges to 0 when the sampling period goes to 0.

Proposition 2. $\lim_{T \to 0} \mathit{err}_T(s, i) = 0$

Intuitively, the quantization step abstracts the real value of $s[T]$ to the nearest multiple of the quantization step. This approximation inevitably introduces an accumulative error to the distance between quantized signals. We provide a bound on this error as a function of Q and the length of the signals, and show that the error bound decreases with smaller quantization steps. We now formalize this result. We first define the WED error due to quantization.

Definition 6 (Weighted Edit Distance Error). *Let $s_1[T]$ and $s_2[T]$ be two discrete signals of length l and Q a quantization step. The WED error, denoted by $\mathit{err}_{\mathsf{Q}}(s_1[T], s_2[T])$, is defined as follows*

$$\mathit{err}_{\mathsf{Q}}(s_1, s_2) = |d_W(s_1[T], s_2[T]) - d_W(s_1[T][\mathsf{Q}], s_2[T][\mathsf{Q}])|$$

The following theorem bounds the WED error due to quantization and states that the error bound improves as the quantization step approaches 0.

Theorem 1. *For arbitrary discrete signals $s_1[T]$ and $s_2[T]$ of length l defined over the same value domain and quantization step Q,*

1. $\mathit{err}_{\mathsf{Q}}(s_1[T], s_2[T]) \leq \mathsf{Q}|X|l$
2. $\lim_{\mathsf{Q} \to 0} \mathit{err}_{\mathsf{Q}}(s_1[T], s_2[T]) = 0$

4 Weighted Edit Robustness for Signal Temporal Logic

In this section, we propose a novel procedure for computing the *robustness degree* of a discrete signal with respect to a STL property. In our approach, we set c_i and c_d to be equal to $|X|(v_{max} - v_{min})$. In other words, the deletion and insertion costs are at most the largest substitution cost. Our procedure relies on computing the WED between a signal and a set of signals, defined by the specification. It consists of several steps, illustrated in Fig. 2. We first translate the STL formula φ into a symbolic automaton $\mathcal{A}_\varphi$ that accepts the same language as the specification. The automaton $\mathcal{A}_\varphi$ treats timing constraints from the formula enumeratively, but keeps symbolic guards on data variables[3]. We then transform $\mathcal{A}_\varphi$ into a *weighted edit automaton* $\mathcal{W}_\varphi$, a weighted symbolic automaton that accepts all the signals but with the value that corresponds to the WED between the signal and the specification (Fig. 2 (a)). We propose an algorithm for computing this distance. Computing the robustness degree between a signal and an STL specification follows from the calculation of their WED, as shown in Fig. 2 (b).

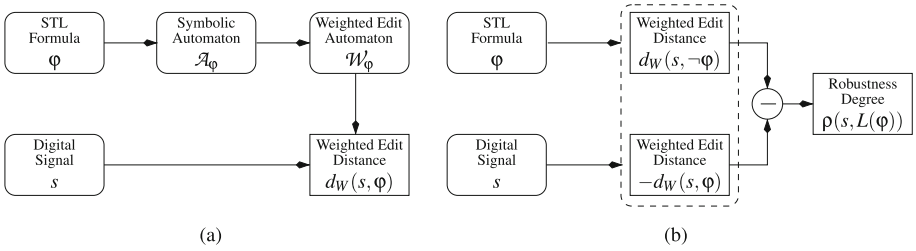

Fig. 2. Computation of (a) $d_W(s,\varphi)$ and $\rho(s,\varphi)$.

4.1 From STL to Weighted Edit Automata

Let X be a set of finite variables defined over the domain $\mathbb{D} = [v_{min}, v_{max}] \subseteq \mathbb{N}$. We consider an STL formula φ defined over X. Let $s : [0, l) \times X \to \mathbb{D}$ be a digital signal.

From φ to $\mathcal{A}_\varphi$. In the first step, we translate the STL specification φ into the automaton $\mathcal{A}_\varphi$ such that $L(\varphi) = L(\mathcal{A}_\varphi)$. The translation from STL interpreted over discrete time and finite valued domains to finite automata is standard, and can be achieved by using for instance on-the-fly tableau construction [14] or the temporal testers approach [21].

Example 1. Consider the past STL formula $\varphi = \Box\,(x = 4 \to \diamondsuit\,(x < 3))$, where x is defined over the domain $[0, 5]$. The resulting automaton $\mathcal{A}_\varphi$ is shown in Fig. 3(a).

[3] The time in $\mathcal{A}_\varphi$ cannot be treated symbolically with digital clocks since every pair of states and clock valuation may behave differently with respect to the WED.

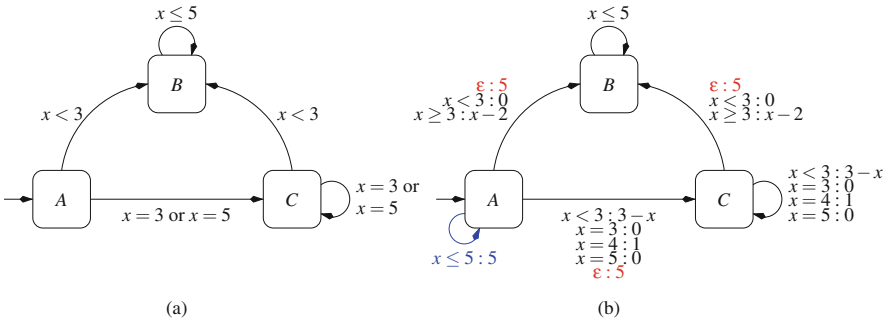

Fig. 3. (a) $\mathcal{A}_\varphi$ accepting $L(\varphi)$ - all states are accepting and (b) $\mathcal{W}_\varphi$. (Color figure online)

From $\mathcal{A}_\varphi$ to $\mathcal{W}_\varphi$. In this step, we translate the automaton $\mathcal{A}_\varphi$ to the weighted edit automaton $\mathcal{W}_\varphi$. The automaton $\mathcal{W}_\varphi$ reads an input signal and mimics the weighted edit operations. In essence, $\mathcal{W}_\varphi$ accepts every signal along multiple paths. Each accepting path induced by the signal corresponds to a sequence of weighted edit operations needed to transform the input signal into another one allowed by the specification. The value of the least expensive path corresponds to the weighted edit distance between the input signal and the specification. The weighted automaton $\mathcal{W}_\varphi$ explicitly treats substitution, insertion and deletion operations, by augmenting $\mathcal{A}_\varphi$ with additional transitions and associating to them the appropriate weight function. We now provide details of the translation and describe the handling of weighted edit operations. Let $\mathcal{A}_\varphi = (\mathbb{D}, X, Q, I, F, \Delta)$ be the symbolic automaton accepting the language of the specification φ.

Substitution. In order to address substitutions in the automaton, we define a new set of *substitution* transitions Δ_s and associate to them the weight function λ_s as follows. Given $q, q' \in Q$, let $\gamma(q, q') = \bigvee_{(q,\gamma,q')\in\Delta} \gamma$. Then, we have:

– $(q, \mathsf{true}, q') \in \Delta_s$ if there exists $(q, \gamma, q') \in \Delta$ for some γ; and
– $\lambda_s((q, \mathsf{true}, q'), v) = d_M(v, \gamma(q, q'))$, for all $v \in \mathbb{D}^{|X|}$.

Intuitively, we replace all the transitions in $\mathcal{A}_\varphi$ with new ones that have the same source and target states. We relax the guards in the new transitions and make them enabled for *any* input. On the other hand, we control the cost of making a transition with the weight function λ_s, which computes the substitution cost needed to take the transition with a specific input. This cost is the Manhattan distance between the input value and the guard associated to the original transition.

Deletion. Addressing deletion operations consists in adding self-loop transitions that consume all the input letters to all the states with the deletion cost $c_d = |X|(v_{max} - v_{min})$, thus mimicking deletion operations. We skip adding a self-loop transition to states that already have the same substitution self-loop transition – according to our definition $c_d \geq c_s(a, X)$ for all a, hence taking the deletion

transition instead of the substitution one can never improve the value of a path and is therefore redundant. We define the set of deletion transitions Δ_d and the associated weight function λ_d as follows:

- $(q, \mathsf{true}, q) \in \Delta_d$ if $(q, \mathsf{true}, q) \notin \Delta_s$; and
- $\lambda_d(\delta, v) = c_d$ for all $\delta \in \Delta_d$ and $v \in \mathbb{D}^{|X|}$.

Insertion. In order to mimic the insertion operations, we augment the transitions relation of $\mathcal{W}_\varphi$ with silent transitions. For every original transition in Δ, we associate another transition with the same source and target states, but labeled with ϵ and having the insertion cost $c_i = |X|(v_{max} - v_{min})$. Formally, we define the set of insertion transitions Δ_i and the associated weight function λ_i as follows:

- $(q, \epsilon, q') \in \Delta_i$ if $(q, \gamma, q') \in \Delta$ for some γ; and
- $\lambda_i(\delta, \{\epsilon\}) = c_i$ for all $\delta \in \Delta_i$.

Given the symbolic automaton $\mathcal{A}_\psi = (\mathbb{D}, X, Q, I, F, \Delta)$ accepting the language of the specification φ, its associated symbolic weigthed edit automaton $\mathcal{W}_\psi$ is the tuple $(\mathbb{D}, X, Q, I, F, \Delta', \lambda')$, where $\Delta' = \Delta_s \cup \Delta_d \cup \Delta_i$ and $\lambda'(\delta, v) = \lambda_s(\delta, v)$ if $\delta \in \Delta_s$, $\lambda'(\delta, v) = \lambda_d(\delta, v)$ if $\delta \in \Delta_d$ and $\lambda'(\delta, \epsilon) = \lambda_i(\delta, \epsilon)$ if $\delta \in \Delta_i$.

Example 2. The weighted edit automaton $\mathcal{W}_\varphi$ obtained from $\mathcal{A}_\varphi$ is illustrated in Fig. 3(b). The blue transitions, such as $(A, 0, A)$ with weight 5, correspond to the deletion transitions. The red transitions, such as (A, ϵ, B), correspond to the insertion transitions.

The resulting weighted automaton $\mathcal{W}_\varphi$ allows determining the weighted edit distance between a signal w and the formula φ, by computing the value of s in $\mathcal{W}_\varphi$.

Theorem 2. $d_W(s, \varphi) = v(s, \mathcal{W}_\varphi)$.

4.2 Computing the Value of a Signal in a Weighted Edit Automaton

We now present an on-the-fly algorithm Val, shown in Algorithm 1, that computes the value of a signal s in a weighted automaton $\mathcal{W}$. In every step i, the algorithm computes the minimum cost of reaching the state q with the prefix of s consisting of its first i values. After reading a prefix of s, we may reach a state $q \in Q$ in different ways with different costs. Note that it is sufficient to keep the state with the minimum value in each iteration. It follows that the algorithm requires book keeping $|Q|$ state value fields in every iteration. We now explain the details of the algorithm. The procedure first initializes the costs of all the states in $\mathcal{W}$ (see Algorithm 2). The initial states are set to 0 and the non-initial ones to ∞. Then, we compute the effect of taking the ϵ transitions without reading any signal value. It is sufficient to iterate this step $|Q|$ times, since within $|Q|$ iterations, one is guaranteed to reach a state q that was already

visited with a smaller value v. In every subsequent iteration i, we first update the state values by applying the cost of taking all transitions labeled by $s(i, X)$ and then update the effect of taking ϵ transitions $|Q|$ times. The weight function of a substitution cost is computed as follows: $\lambda(v, x \leq k)$ gives 0 if $v \leq k$, and $v - k$ otherwise; $\lambda(v, \neg(x \leq k))$ is symmetric; $\lambda(v, \varphi_1 \wedge \varphi_2) = \max(\lambda(v, \varphi_1), \lambda(v, \varphi_2))$ and $\lambda(v, \varphi_1 \vee \varphi_2) = \min(\lambda(v, \varphi_1), \lambda(v, \varphi_2))$.

Upon termination, the algorithm returns the minimum cost of reaching an accepting state in the automaton.

Theorem 3. $\mathit{Val}(s, \mathcal{W}) = v(s, \mathcal{W})$.

Theorem 4. *Given a signal s of length l defined over X and a weighted automaton $\mathcal{W}$ with n states and m transitions, $\mathit{Val}(s, \mathcal{W})$ takes in the order of $O(lnm))$ iterations to compute the value of s in $\mathcal{W}$, and requires in the order of $O(n(\lceil log(l(v_{\max} - v_{\min}))\rceil))$ memory.*

Algorithm 1. $\mathsf{Val}(s, \mathcal{W})$

Input: s and $\mathcal{W}_\psi$
Output: v
  $\mathsf{InitVal}(\mathcal{W})$
  for all $i \in [0, l)$ **do**
    for all $\delta = (q, \gamma, q') \in \Delta$ **do**
      $v'(q') \leftarrow \min(v'(q'), v(q) + \lambda(s(i, X), \delta))$
    end for
    for $i = 0; i < |Q|; i++$ **do**
      for all $\delta = (q, \epsilon, q') \in \Delta$ **do**
        $v'(q') \leftarrow \min(v'(q'), v(q) + \lambda(\delta, \epsilon))$
      end for
      for all $q \in Q$ **do**
        $v(q) \leftarrow v'(q)$
        $v'(q) \leftarrow \infty$
      end for
    end for
  end for
  $v \leftarrow \min_{q \in F} v(q)$
    return v

Algorithm 2. $\mathsf{InitVal}(\mathcal{W})$

for all $q \in Q$ **do**
  $v(q) \leftarrow (q \in I)\ ?\ 0 : \infty; v'(q) \leftarrow \infty$
end for
for $i = 0; i < |Q|; i++$ **do**
  for all $\delta = (q, \epsilon, q') \in \Delta$ **do**
    $v'(q') \leftarrow \min(v'(q'), v(q) + \lambda(\delta, \epsilon))$
  end for
  for all $q \in Q$ **do**
    $v(q) \leftarrow v'(q)$
    $v'(q) \leftarrow \infty$
  end for
end for

Example 3. Consider the STL property φ from Example 1, the associated weighted edit automaton $\mathcal{W}_\varphi$ from Fig. 1 and the signal[4] $s : [0, 2] \to [0, 5]$ such that $s(0) = 5$, $s(1) = 5$ and $s(2) = 4$. It is clear that $(s, 0) \not\models \varphi$, since $s(2) = 4$, while there was not a single $0 \leq i < 2$ where $s(i) < 3$. We illustrate in Fig. 4 the computation of $v(s, \mathcal{W}_\varphi)$. We can see that with the signal s, we can reach one of the accepting states (B or C) with the value 1. This value corresponds to one substitution operation, replacing the value of 4 in $s(2)$ by 5, which allows vacuous satisfaction of the property φ.

[4] Since s has only one component, we skip the variable name.

		$s(0)=5$		$s(1)=5$		$s(2)=4$		
A	0	0	5	5	10	10	15	15
B	∞	5	3	3	3	3	3	3
C	∞	5	0	0	0	0	1	1
	init	ε-init	update	ε-update	update	ε-update	update	ε-update

Fig. 4. Example - computation of $v(s, \mathcal{W}_\varphi)$.

5 Implementation and Case Study

We now describe our implementation of quantitative monitors for STL. The parser for the STL formulas is developed using Java and ANTLR. We translate STL properties into temporal testers, and convert them to acceptor automata. We use JAutomata library to represent the testers and the acceptors. We then generate quantitative monitor code in Verilog HDL. The resulting monitor is a hardware implementation of the weighted automata and the underlying algorithm for computing the weighted edit distance.

Table 1. Automatic transmission properties [4].

ID	φ
φ_1	$\square\ (\omega < 4500)$
φ_2	$\square\ ((\omega < 4500) \wedge (v < 120))$
φ_3	$\square\ ((g_2 \wedge \bigcirc\ g_1) \rightarrow \square_{(0,2.5]}\ \neg g_2)$
φ_4	$\square\ ((\neg g_1 \wedge \bigcirc\ g_1) \rightarrow \square_{(0,2.5]}\ g_1)$
φ_5	$\bigwedge_{i=1}^{4} \square\ ((\neg g_i \wedge \bigcirc\ g_i) \rightarrow \square_{(0,2.5]}\ g_i)$
φ_6	$\neg(\diamondsuit_{[0,4]}\ (v > 120) \wedge \square\ (\omega < 4500))$
φ_7	$\diamondsuit_{[0,4]}\ ((v > 120) \wedge \square\ (\omega < 4500))$
φ_8	$((g_1\, \mathcal{U} g_2\, \mathcal{U} g_3\, \mathcal{U} g_4) \wedge \diamondsuit_{[0,10]}\ (g_4 \wedge \diamondsuit_{[0,2]}\ (\omega > 4500))) \rightarrow \diamondsuit_{[0,10]}\ (g_4 \rightarrow \bigcirc\ (g_4\, \mathcal{U}_{[0,1]} (v \geq 120)))$

For the evaluation of our approach, we apply it to an automotive benchmark problem published in [4]. We consider the slightly modified Automatic Transmission deterministic Simulink demo provided by Mathworks as our system-under-test (SUT). It is a model of an automatic transmission controller that exhibits both continuous and discrete behavior. The system has two inputs – the throttle u_t and the break u_b. The break allows the user to model variable load on the engine. The system has two continuous-time state variables – the speed of the

engine ω (RPM), the speed of the vehicle v (mph) and the active gear g_i. The system is initialized with zero vehicle and engine speed. It follows that the output trajectories depend only on the input signals u_t and u_b, which can take any value between 0 and 100 at any point in time. The Simulink model contains 69 blocks including 2 integrators, 3 look-up tables, 2 two-dimensional look-up tables and a Stateflow chart with 2 concurrently executing finite state machines with 4 and 3 states, respectively. The benchmark defines 8 STL formalized requirements that the system shall satisfy, shown in Table 1.

We now describe the evaluation setup. We simulated the Simulink model with fixed-step sampling and recorded the results. The obtained traces, as the one shown in Fig. 5, were then further discretized with the uniform quantization. We have obtained 751 samples from the Simulink model and normalized all variables' value domain to the interval $[0, 5000]$ which is the range of RPM variable, thus achieving fair reasoning about their substitution cost. We designed a test-bench in Verilog to stimulate the monitor with the generated values from the Simulink model. We used Xilinx Vivado to perform monitor simulation and synthesis.

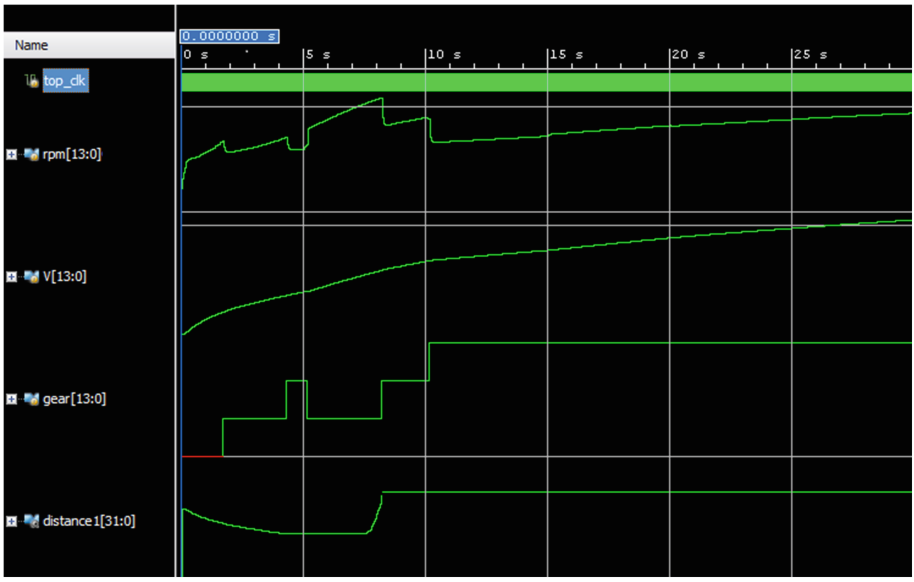

Fig. 5. A simulation trace s from the Automatic Transmission model and $d_W(s, \neg \varphi_6)$.

Figure 5 illustrates the monitoring results for φ_6 on a specific gear input. In the depicted scenario, the speed does not reach 120 mph in 4 s, a sufficient condition for the satisfaction of the formula. In order to violate the formula, we need to alter both v and ω signals such that (1) v reaches 120 mph at any moment within the first 4 s; and (2) ω remains continuously below 4500 rpm. These alterations result in (1) a single substitution happening within the first

Table 2. Evaluation results.

φ	ρ	φ				$\neg\varphi$			
		$\lvert Q\rvert$	$\lvert\Delta\rvert$	#FF	#LUT	$\lvert Q\rvert$	$\lvert\Delta\rvert$	#FF	#LUT
φ_1	-2528	2	2	62	260	4	8	94	657
φ_2	-11423	2	2	75	306	4	11	107	799
φ_3	1000	496	1374	4106	53033	992	2878	8127	106937
φ_4	1000	496	692	3061	22777	992	1445	6025	44968
φ_5	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
φ_6	5337	405	813	6540	66085	409	903	6504	73657
φ_7	-5336	403	903	6504	73766	405	813	6545	66116
φ_8	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a

4 s which is necessary to bring v to 120 mph; and (2) the accumulation of substitution costs in the interval between 7 and 8 s of the simulation where ω actually exceeds 4500 rpm. Note that the robustness degree decreases in the first 4 s. This happens because the actual v increases and the substitution cost needed for v to reach 120 mph is continuously being improved.

The evaluation results are shown in Table 2. We tested the correctness of STL to automata translation by generating both acceptors for φ and $\neg\varphi$. The presented robustness degrees are not normalized, which can be statically computed using the formula from Sect. 3. It is clear from our table that either the distance from φ or from its negation is always 0. The dominant type of resources when implementing our monitors on FPGA hardware are LUTs. This is not surprising, due to the large combinatorial and arithmetic requirements of the computation. We can also note that the size of our monitors is sensitive to the timing bounds in the formulas and the sampling period of the input signals. Our monitor automata enumerate clock ticks instead of using a symbolic representation. The enumeration is necessary because state - clock valuation pairs can have different values associated and thus cannot be grouped. We were not able to generate monitors for φ_5 and φ_8 due to the state explosion. However, φ_5 can be decomposed into 4 independent sub-properties. We can see several ways to handle large properties such as φ_8 that we will investigate in the future – by reformulating the specification using both past and future operators, by using larger sampling periods (and thus smaller time bounds in the formula) and by using more powerful FPGA hardware.

6 Conclusions

In this paper, we proposed a new procedure for measuring robustness of STL properties based on the weighted edit distance. Weighted edit distance is an accumulative measure, and provides insight on how often the property is violated.

It is thus sensitive to the length of the signal, but also to the sampling rate and the number of components in the signal. Normalizing the distance enables obtaining a uniform measure of "goodness" of a behavior. While the focus is on the quantitative semantics of STL, our approach is applicable to other regular specification languages interpreted over finite signals.

In the future, we will study more closely the effect of sampling to the computed distance. We will apply our approach on more relevant examples, in order to get better insight on the interpretation of the values obtained by the robustness measurements. Finally, we will work on the optimization of our procedure in order to obtain smaller weighted automata that can be efficiently implemented on hardware, and will deploy our implementation on FPGA and evaluate it in the lab environment.

Acknowledgements. We would like to thank Oded Maler, Mario Klima and the anonymous reviewers for their comments on the earlier drafts of the paper.

We acknowledge the support of the IKT der Zukunft of Austrian FFG project HARMONIA (nr. 845631), the ICT COST Action IC1402 Runtime Verification beyond Monitoring (ARVI), the Austrian National Research Network S 11405-N23 and S 11412-N23 (RiSE/SHiNE) of the Austrian Science Fund (FWF) and the Doctoral Program Logical Methods in Computer Science of the Austrian Science Fund (FWF).

References

1. Akazaki, T., Hasuo, I.: Time robustness in MTL and expressivity in hybrid system falsification. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9207, pp. 356–374. Springer, Heidelberg (2015). doi:10.1007/978-3-319-21668-3_21
2. Allauzen, C., Mohri, M.: Linear-space computation of the edit-distance between a string and a finite automaton. CoRR abs/0904.4686 (2009)
3. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TaLiRo: a tool for temporal logic falsification for hybrid systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 254–257. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19835-9_21
4. Abbas, H., Hoxha, B., Fainekos, G.: Benchmarks for temporal logic requirements for automotive systems. In: Proceedings of Applied Verification for Continuous and Hybrid Systems (2014)
5. Bartocci, E., Bortolussi, L., Sanguinetti, G.: Data-driven statistical learning of temporal logic properties. In: Legay, A., Bozga, M. (eds.) FORMATS 2014. LNCS, vol. 8711, pp. 23–37. Springer, Heidelberg (2014). doi:10.1007/978-3-319-10512-3_3
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. Deshmukh, J.V., Majumdar, R., Prabhu, V.S.: Quantifying conformance using the Skorokhod metric. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9207, pp. 234–250. Springer, Heidelberg (2015). doi:10.1007/978-3-319-21668-3_14
8. Deshmukh, J.V., Majumdar, R., Prabhu, V.S.: Quantifying conformance using the Skorokhod metric (full version). CoRR abs/1505.05832 (2015)
9. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 167–170. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14295-6_17

10. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for STL. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 264–279. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39799-8_19
11. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 92–106. Springer, Heidelberg (2010). doi:10.1007/978-3-642-15297-9_9
12. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for continuous-time signals. Theor. Comput. Sci. **410**(42), 4262–4291 (2009)
13. Fainekos, G.E., Sankaranarayanan, S., Ivancic, F., Gupta, A.: Robustness of model-based simulations. In: Proceedings of the 30th IEEE Real-Time Systems Symposium, RTSS 2009, Washington, DC, USA, 1–4 December 2009, pp. 345–354 (2009)
14. Gerth, R., Peled, D., Vardi, M.Y., Wolper, P.: Simple on-the-fly automatic verification of linear temporal logic. In: Protocol Specification, Testing and Verification XV, Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification, Warsaw, Poland, pp. 3–18 (1995)
15. Konstantinidis, S.: Computing the edit distance of a regular language. Inf. Comput. **205**(9), 1307–1316 (2007)
16. Krause, E.F.: Taxicab Geometry: An Adventure in Non-Euclidean Geometry. Courier Corporation, North Chelmsford (2012)
17. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, reversals. Sov. Phys. Dokl. **10**, 707 (1966)
18. Maler, O., Nickovic, D.: Monitoring properties of analog and mixed-signal circuits. STTT **15**(3), 247–268 (2013)
19. Mohri, M.: Edit-distance of weighted automata: general definitions and algorithms. Int. J. Found. Comput. Sci. **14**(6), 957–982 (2003)
20. Nguyen, T., Ničković, D.: Assertion-based monitoring in practice–checking correctness of an automotive sensor interface. In: Lang, F., Flammini, F. (eds.) FMICS 2014. LNCS, vol. 8718, pp. 16–32. Springer, Heidelberg (2014). doi:10.1007/978-3-319-10702-8_2
21. Pnueli, A., Zaks, A.: On the merits of temporal testers. In: Grumberg, O., Veith, H. (eds.) 25 Years of Model Checking. LNCS, vol. 5000, pp. 172–195. Springer, Heidelberg (2008). doi:10.1007/978-3-540-69850-0_11
22. Rizk, A., Batt, G., Fages, F., Soliman, S.: On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology. In: Heiner, M., Uhrmacher, A.M. (eds.) CMSB 2008. LNCS (LNBI), vol. 5307, pp. 251–268. Springer, Heidelberg (2008). doi:10.1007/978-3-540-88562-7_19
23. Samanta, R., Deshmukh, J.V., Chaudhuri, S.: Robustness analysis of string transducers. In: Van Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 427–441. Springer, Heidelberg (2013). doi:10.1007/978-3-319-02444-8_30
24. Schulz, K.U., Mihov, S.: Fast string correction with Levenshtein automata. Int. J. Doc. Anal. Recogn. **5**(1), 67–85 (2002)
25. Veanes, M., Bjørner, N., de Moura, L.: Symbolic automata constraint solving. In: Fermüller, C.G., Voronkov, A. (eds.) LPAR-17. LNCS, vol. 6397, pp. 640–654. Springer, Heidelberg (2010). doi:10.1007/978-3-642-16242-8_45
26. Wagner, R.A.: Order-n correction for regular languages. Commun. ACM **17**(5), 265–268 (1974)