2018 IEEE Conference on Decision and Control (CDC)
Miami Beach, FL, USA, Dec. 17-19, 2018

Metrics for Signal Temporal Logic Formulae

Curtis Madsen*, Prashant Vaidyanathan*, Sadra Sadraddini*, Cristian-Ioan Vasile*, Nicholas A. DeLateur, Ron Weiss, Douglas Densmore, and Calin Belta

Abstract— **Signal Temporal Logic (STL) is a formal language for describing a broad range of real-valued, temporal properties in cyber-physical systems. While there has been extensive research on verification and control synthesis from STL requirements, there is no formal framework for comparing two STL formulae. In this paper, we show that under mild assumptions, STL formulae admit a metric space. We propose two metrics over this space based on i) the Pompeiu-Hausdorff distance and ii) the symmetric difference measure and present algorithms to compute them. Alongside illustrative examples, we present an application of these metrics as *design quality measures* where they are used to compare all the temporal behaviors of a designed system, such as a synthetic genetic circuit, with the "desired" specification.**

I. Introduction

Temporal logics [1] are increasingly used for describing specifications in cyber-physical systems such as robotics [2], synthetic biology [3], and transportation [4]. Variants of temporal logics, such as *Signal Temporal Logic* (STL) [5], can naturally describe a wide range of temporal system properties such as *safety* (never visit a "bad" state), *liveness* (eventually visit a "good" state), *sequentiality*, and their arbitrarily elaborate combinations.

Using *model checking* [1] techniques, signals or traces can be checked to determine whether or not they satisfy a specification. For STL in particular, the *degree of satisfaction* or *robustness* is a quantitative measure to characterize how far a signal is from satisfaction [6], [7] of an STL formula. There is currently, however, no formal way to directly compare specifications against each other. Previous related approaches in planning and control have looked into specification relaxation, where the goal is to minimally enlarge the specification language to include a satisfying control policy for the system model. Various specification relaxations have been defined including minimum violation [8] for self-driving cars, temporal relaxation of deadlines [9], minimum revision of Büchi automata [10], and diagnosis and repair in reactive synthesis [11]. While language inclusion and equivalence problems are of paramount importance in computer science and control theory, they are only qualitative measures while we are interested in quantitative metrics.

This paper presents two metrics that can be used to compute the distance between two STL specifications. Under mild assumptions, we propose metrics based on the languages of STL formulae. We propose two distance functions. The first is based on the *Pompeiu-Hausdorff* (PH) distance [12], which captures how much the language of one formula must be enlarged to include the other, and the second is based on the *symmetric difference* (SD) [13], which characterizes how much overlap there is between the two formulae. The theoretical contributions of this paper are: 1) formalization of STL formulae metrics based on the PH and the SD distances, and 2) methods for computing the PH using *mixed-integer linear programming* (MILP), and the SD using a recursive algorithm based on the *area of satisfaction*. We discuss the comparison of the two metrics in detail and provide examples that highlight their differences.

This paper additionally presents an application of the proposed metrics to a behavioral synthesis problem. We are interested in generating designs that exhibit desired behaviors specified in STL. For example, we study synthetic genetic circuits. Possible circuit designs are constructed and measured in laboratory experiments, and the resulting traces are abstracted into STL specifications using *temporal logic inference* (TLI). These formulae are compared quantitatively against the desired design specification using the proposed metrics. The related contribution is a design quality measure for evaluating implementations against STL specifications. [1]

II. Preliminaries

Let $\mathbb{R}, \mathbb{R}_{\geq 0}$, $\mathbb{N}$ denote the set of real, non-negative real, and natural numbers, respectively. The absolute value of $r \in \mathbb{R}$ is denoted by $|r|$, and the infinity-norm of $x \in \mathbb{R}^n$ by $\|x\|_\infty := \max_{i\in\{1,\cdots,n\}} |x^i|$, where x^i is the i'th component of x. A function $f : \mathbb{R}^n \to \mathbb{R}$ is *rectangular* if $f(x) = x^i$, $i \in \{1, \cdots, n\}$. A metric space $(\mathcal{M}, d)$ is composed of a set $\mathcal{M}$ and a distance $d : \mathcal{M} \times \mathcal{M} \to \mathbb{R}_{\geq 0}$. We use discrete time throughout the paper. Time intervals $I = [t_1, t_2]$, $t_1, t_2 \in \mathbb{N}, t_1 \leq t_2$, are interpreted as $\{t_1, t_1 + 1, \cdots, t_2\}$, and $\tau + I = [\tau + t_1, \tau + t_2]$, $\tau \in \mathbb{N}$. The continuous interval $\{r | 0 \leq r \leq 1\}$ is denoted by $\mathbb{U}$. An n-dimensional, real, infinite-time, discrete-time signal s is a string $s_0 s_1 s_2 \cdots$, where $s_t \in \mathbb{S} \subset \mathbb{R}^n$, $t \in \mathbb{N}$. The *suffix* $s[t]$ of s at t is a signal such that $s[t]_\tau = s_{t+\tau}$, $\forall t, \tau \in \mathbb{N}$, and $s[t_1, t_2] := s_{t_1} s_{t_1+1} \cdots s_{t_2}$. The set of all signals with values in $\mathbb{S}$ is $\mathcal{S}$. The set of all signal prefixes with time bound T is $\mathcal{S}_T :=$

*These authors contributed equally. This work was partially supported by the National Science Foundation under grant CPS Frontier 1446607 and NSF IIS-1723995. Curtis Madsen (ckmadsen@bu.edu), Prashant Vaidyanathan (prash@bu.edu), Douglas Densmore (dougd@bu.edu), and Calin Belta (cbelta@bu.edu) are at Boston University, Boston, MA, USA. Sadra Sadraddini (sadra@mit.edu), Cristian-Ioan Vasile (cvasile@mit.edu), Nicholas A. DeLateur (delateur@mit.edu), and Ron Weiss (rweiss@mit.edu) are at Massachusetts Institute of Technology, Cambridge, MA, USA.

[1]The Extended version of this paper is available at `https://arxiv.org/abs/1808.03315`. The software is publicly accessible at `https://github.com/CIDARLAB/stl_metrics`.

$\{s[0,T] \mid s \in \mathcal{S}\}$. For the convenience of notation, we use $s \in \mathcal{S}_T$ to say $s[0,T] \in \mathcal{S}_T$. The distance between two signals $s, s' \in \mathcal{S}_T$ is $d(s,s') := \sup_{t\in[0,T]} \{\|s_t - s'_t\|_\infty\}$.

Signal Temporal Logic: The syntax of STL is [5]:

$$\phi ::= \top \mid \pi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \mathcal{U}_I \phi_2 \ ,$$

where $\top$ is the Boolean *true* constant; π is a predicate over $\mathbb{R}^n$ in the form of $f(x) \sim \mu$, $f : \mathbb{S} \to \mathbb{R}$, $\mu \in \mathbb{R}$, and $\sim \in \{\leq, \geq\}$; $\neg$ and $\wedge$ are the Boolean operators for negation and conjunction, respectively; and $\mathcal{U}_I$ is the temporal operator *until* over bounded interval I. A predicate $f(s) \sim \mu$ is rectangular if f is rectangular. Other Boolean operations are defined in the usual way. Additional temporal operators *eventually* and *globally* are defined as $\Diamond_I \phi \equiv \top \mathcal{U}_I \phi$ and $\Box_I \phi \equiv \neg \Diamond_I \neg \phi$, respectively, where I is an interval. The set of all STL formulae over signals in $\mathcal{S}$ is denoted by $\Phi^{\mathcal{S}}$. The *STL score*, also known as *robustness degree* is a function $\rho : \mathcal{S} \times \Phi^{\mathcal{S}} \times \mathbb{N} \to \mathbb{R}$, which is recursively defined as [5]:

$$\begin{aligned}
\rho(s, f(s) \sim \mu, t) \quad &= \begin{cases} \mu - f(s_t) & \sim = \leq \\ f(s_t) - \mu & \sim = \geq \end{cases}, \\
\rho(s, \neg\phi, t) \quad &= -\rho(s, \phi, t), \\
\rho(s, \phi_1 \wedge \phi_2, t) \quad &= \min(\rho(s, \phi_1, t), \rho(s, \phi_2, t)), \\
\rho(s, \phi_1 \ \mathcal{U}_I \ \phi_2, t) \quad &= \max_{t' \in t+I} \big(\rho(s, \phi_2, t'), \\
&\qquad \min_{t'' \in [t,t']} \rho(s, \phi_1, t'')\big),
\end{aligned} \tag{1}$$

As one can inspect from (1), a signal *satisfies* an STL specification at a certain time if and only if its corresponding STL score is positive: $s[t] \models \phi \Leftrightarrow \rho(s, \phi, t) > 0$, where $\models$ is read as "satisfies". The case of $\rho = 0$ is usually left ambiguous - this is never a concern in practice due to issues with numerical precision. In this paper, we consider $\rho = 0$ as satisfaction, but by doing so, we sacrifice the principle of contradiction: $s[t] \models \phi$ and $s[t] \models \neg\phi$ if $\rho(s, \phi, t) = 0$.

The horizon of an STL formula is defined as the minimum length of the time window required to compute its score [14]:

$$\begin{aligned}
&\|\pi\| = 0, \|\phi\| = \|\neg\phi\|, \|\phi_1 \wedge \phi_2\| = \max\{\|\phi_1\|, \|\phi_2\|\} \\
&\|\phi_1 \mathcal{U}_{[t_1,t_2]} \phi_2\| = t_2 + \max\{\|\phi_1\|, \|\phi_2\|\}
\end{aligned} \tag{2}$$

The set of all STL formulae over $\mathcal{S}$ such that their horizons are less than T is denoted by $\Phi^{\mathcal{S}_T}$. Note that computing $\rho(s, \phi, t)$ requires $s[t : t + \|\phi\|]$, and the rest of the values are irrelevant. Given $\phi \in \Phi^{\mathcal{S}_T}$, we define the bounded-time language as: $\mathcal{L}(\phi) := \{s \in \mathcal{S}_T \mid \rho(s, \phi, 0) \geq 0\}$. Note that $\mathcal{L}(\phi) \subset \mathbb{R}^{n(T+1)}$. With rectangular predicates, the bounded-time language becomes a finite union of hyper-rectangles.

Example 1: Let $\mathbb{S} = \mathbb{U}$, and the STL formulae in $\Phi^{\mathcal{S}_{20}}$:

$$\begin{aligned}
\phi_1 = \Box_{[0,20]}\theta_1, \phi_2 = \Box_{[0,20]}\theta_2, \\
\phi_3 = \Diamond_{[0,20]}\theta_1, \phi_4 = \Box_{[0,20]}\theta_1 \wedge \Diamond_{[0,20]}\theta_2, \\
\phi_5 = (\Box_{[0,10]}\theta_1) \wedge (\Box_{[12,20]}\theta_2), \phi_6 = \Box_{[0,16]}\Diamond_{[0,4]}\theta_1
\end{aligned} \tag{3}$$

where $\theta_1 = (x \geq 0.2) \wedge (x \leq 0.4)$, and $\theta_2 = (x \geq 0.2) \wedge (x \leq 0.44)$. We have $\|\phi_i\| = 20, i = 1, \cdots, 6$. Two examples of bounded-time languages are: $\mathcal{L}(\phi_2) = \bigcap_{\tau=0}^{20}\{0.2 \leq x_t \leq 0.44\}, \mathcal{L}(\phi_3) = \bigcup_{\tau=0}^{20}\{0.2 \leq x_t \leq 0.4\}$. Consider two constant signals s^1 and s^2, where $s^1_t = 0.3, s^2_t = t/20, t = 0, 1, \cdots, 20$. The STL scores are computed from (1). For instance, $\rho(s^1, \phi_1, 0) = 0.1$, $\rho(s^2, \phi_1, 0) = -0.6$ (minimizer at $t = 20$), and $\rho(s^2, \phi_3, 0) = 0.1$ (maximizer at $t = 6$).

III. Metrics

In this section, we introduce two functions $d_{STL} : \Phi^{\mathcal{S}_T} \times \Phi^{\mathcal{S}_T} \to \mathbb{R}_{\geq 0}$ that quantify the dissimilarity between the properties captured by the two STL formulae. However, it *is* possible that different formulae may describe the same properties. For example, ϕ_1 and ϕ_4 in (3) are describing the same behavior, since any signal that satisfies ϕ_1 already satisfies ϕ_4 *and* vice versa. The key idea is to define the distance between two STL formulae as the distance between their time-bounded languages.

Assumption 1: The set $\mathbb{S} \subset \mathbb{R}^n$ is compact.

Assumption 2: All of the predicates are rectangular.

Bounded-time languages are constructed in finite-dimensional Euclidean spaces. Also, since all inequalities in the predicates are non-strict, bounded-time languages are compact sets. Assumption 2 is theoretically restrictive, but not in most applications - usually it is the case that all predicates are rectangular as they describe thresholds for state components of a system.

Definition 1: We say that the two STL formulae ϕ_1 and ϕ_2 are *semantically equivalent*, denoted by $\phi_1 \equiv \phi_2$, if both induce the same language: $\mathcal{L}(\phi_1) = \mathcal{L}(\phi_2)$.

The set of equivalence classes of $\Phi^{\mathcal{S}_T}$ induced by $\equiv$ is denoted by $\Phi^{\mathcal{S}_T}/\equiv$. Distance functions d_{STL} are effectively pseudo-metrics on $\Phi^{\mathcal{S}_T}$, but proper metrics on $\Phi^{\mathcal{S}_T}/\equiv$, where $d_{STL}(\langle\phi_1\rangle, \langle\phi_2\rangle) = d_{STL}(\phi_1, \phi_2)$ is the induced metric, $\langle\phi\rangle$ is the equivalence class associated with ϕ, and ϕ_1 and ϕ_2 are formulae in the two equivalence classes. By definition, there is a one-to-one map between the equivalence classes of STL formulae and their formulae. Moreover, for any $\phi_1, \phi_2 \in \langle\phi\rangle$, we have $d_{STL}(\phi_1, \phi_2) = 0$.

We adapt two common metrics between sets: (a) the *Pompeiu-Hausdorff* (PH) *distance* based on the underlying metric between signals, and (b) a measure of *symmetric difference* (SD) between sets. As it will be clarified in the paper, the choice of T, as long as it is larger than the horizons of the formulae that are considered, does not affect the fundamental properties of the defined metrics. In the case of the PH distance, it does not have any effect at all. For the SD metric, the computed distances are scaled with respect to the inverse of T. These details are explained in Section III-B.

A. Pompeiu-Hausdorff Distance

Definition 2: The (undirected) PH distance is defined as:

$$d_{PH}(\phi_1, \phi_2) = \max\left\{\vec{d}_{PH}(\phi_1, \phi_2), \vec{d}_{PH}(\phi_2, \phi_1)\right\}, \tag{4}$$

where $\vec{d}_{PH}$ denotes the directed PH distance:

$$\vec{d}_{PH}(\phi_1, \phi_2) := \sup_{s_1 \in \mathcal{L}(\phi_1)} \left\{\inf_{s_2 \in \mathcal{L}(\phi_2)} d(s_1, s_2)\right\}. \tag{5}$$

The directed PH distance is obviously not a metric as it is possible to have $\vec{d}_{PH}(\phi_1, \phi_2) \neq \vec{d}_{PH}(\phi_2, \phi_1)$. We have $\vec{d}_{PH}(\phi_1, \phi_2) = 0$ if and only if $\mathcal{L}(\phi_1) \subseteq \mathcal{L}(\phi_2)$. Another way to interpret the PH distance is as follows [12]:

$$\vec{d}_{PH}(\phi_1, \phi_2) = \min\{\epsilon \mid \mathcal{L}(\phi_1) \subseteq \mathcal{L}(\phi_2) + \epsilon\mathcal{B}^{\mathcal{S}_T}\}, \tag{6}$$

where $\mathcal{B}^{\mathcal{S}_T}$ is the unit ball in $\mathcal{S}_T : \{s[0:T] \mid \|s_t\|_\infty \leq 1, t \in [0,T]\}$, and addition of sets is interpreted in the Minkowski sense - $\vec{d}_{PH}(\phi_1, \phi_2)$ is the radius of the minimal ball that should be added to $\mathcal{L}(\phi_2)$ such that it contains $\mathcal{L}(\phi_1)$.

Proposition 1: The $(\Phi^{\mathcal{S}_T}/\equiv, d_{PH})$ is a metric space.

It is possible to interpret (5) as the distance between an STL formula and a signal: $\vec{d}_{PH}(s, \phi) := \min_{s' \in \mathcal{L}(\phi)} d(s, s')$. It is easy to see that we have $\vec{d}_{PH}(s, \phi) = 0$ if and only if $s \in \mathcal{L}(\phi)$. The following result is a reformulation of Definition 23 in [6], which establishes a connection between the STL score and the notion of *signed distance*.

Proposition 2: Given any $\phi \in \Phi^{\mathcal{S}_T}$ and $s \in \mathcal{S}_T$, the STL score is a signed distance in the sense that:

$$\rho(s, \phi, 0) = \begin{cases} -\vec{d}_{PH}(s, \phi) & \vec{d}_{PH}(s, \phi) > 0, \\ \vec{d}_{PH}(s, \neg\phi) & \vec{d}_{PH}(s, \phi) = 0. \end{cases} \quad (7)$$

The following results are extensions of classical results for signed distances [15].

Corollary 1: For any given two formulae $\phi_1, \phi_2 \in \Phi^{\mathcal{S}_T}$ and a signal $s \in \mathcal{S}_T$, we have the following inequalities:

$$\big| |\rho(s, \phi_1, 0)| - |\rho(s, \phi_2, 0)| \big| \leq d_{PH}(\phi_1, \phi_2)$$
$$\big| \rho(s, \phi_1, 0) - \rho(s, \phi_2, 0) \big| \leq d_{PH}(\phi_1, \phi_2) + d_{PH}(\neg\phi_1, \neg\phi_2)$$

Corollary 2: Given $\epsilon > 0$, define ϵ-neighborhood of an STL formula ϕ as $\{\phi\}_\epsilon = \{\phi' \in \Phi^{\mathcal{S}_T} | d_{PH}(\phi, \phi') \leq \epsilon\}$. Then, $\rho(s, \phi, 0) \geq \epsilon$ implies that $s \models \phi', \forall \phi' \in \{\phi\}_\epsilon$.

B. Symmetric Difference

The SD is denoted by $\triangle$, and defined as $X \triangle Y = (X \setminus Y) \cup (Y \setminus X)$, where X and Y are two sets. It induces a distance between compact sets as the measure of the SD [13].

Definition 3: The SD metric is defined as:

$$d_{SD}(\varphi_1, \varphi_2) = \frac{1}{T+1} |\mathcal{L}(\varphi_1) \triangle \mathcal{L}(\varphi_2)|,$$

where $|\cdot|$ is the Lebesgue measure.

Proposition 3: The $(\Phi^{\mathcal{S}_T}/\equiv, d_{SD})$ is a metric space.

We define the coverage of signal sets in the space-time value set. Formally, we have the map $\mathcal{P} : 2^{\mathcal{S}_T} \to \mathbb{S} \times T\mathbb{U}$ such that $\mathcal{P}(S) = \bigcup_{s \in S} \bigcup_{t \in [0,T]} \{(s_t, \tau) \mid t \leq \tau \leq t+1\}$, where $S \subseteq \mathcal{S}$. For an STL formula ϕ, $\mathcal{P}(\phi) = \mathcal{P}(\mathcal{L}(\phi))$.

Let $\mathcal{S} = \mathbb{U}^n$, and $p = x_i \leq \mu$ be a rectangular predicate with $\mu \in \mathbb{U}$ and $i \in \{1, \ldots, n\}$. The coverage of p is $\mathcal{P}(p) = \big((\mathbb{U}^{i-1} \times \mu\mathbb{U} \times \mathbb{U}^{n-i}) \times \mathbb{U}\big) \cup (\mathbb{U}^n \times \{1 \leq t \leq T\})$.

Theorem 1: If ϕ_1 and ϕ_2 are two STL formulae with the same language, then they cover the same space. Formally, we have $\mathcal{L}(\phi_1) = \mathcal{L}(\phi_2)$ implies $\mathcal{P}(\phi_1) = \mathcal{P}(\phi_2)$.

However, the converse is not true in general. In particular, it can fail for formulae containing disjunctions.

C. Comparison

While the PH distance and the SD difference are both metrics, they have quite different behaviors. Here, we elaborate on these differences and show an illustrative example.

Informally, the PH distance has a stronger *spatial* notion, and is closely connected to STL score, as stated in Corollary 1 and Corollary 2. On the other hand, the SD is a more *temporal* notion, as the areas also capture the length of temporal operators. It is possible that two STL formulae have a large PH distance, but a small SD distance, and vice versa. In applications, the choice is dependent on the user. The most useful may be a convex combination - which is a metric by itself - with a user-given convex coefficient.

Example 2: Consider the six STL formulae in (3). We compute the PH and the SD distances between all pairs of formulae using the methods proposed in Section IV. The lower-half of the table shows the directed PH distances, where the values in i'th row and j'th column are $\vec{d}_{PH}(\phi_i, \phi_j)/\vec{d}_{PH}(\phi_j, \phi_i)$ - the PH distance is the maximum of the two numbers, e.g., $d_{PH}(\phi_1, \phi_2) = 0.04$. We have also included the truth constant in the distance table. It is observed that $\vec{d}_{PH}(\phi_i, \top) = 0, \forall i \in \{1, \cdots, 6\}$, which implies the fact that the language of each formula is contained within the language of $\top$, which is the set of all signals. The opposite direction, $\vec{d}_{PH}(\top, \phi) = d_{PH}(\phi, \top)$, is, informally, the quantification of how *restrictive* ϕ is. It is observed that most values are either $0.6 = \max(1-0.4, 0.2-0)$ or $0.56 = \max(1-0.44, 0.2-0)$, which correspond to the extreme signal that one language contains but the other does not, or it is 0, indicating that one language is a subset of another. For instance, ϕ_3 is a "weak" specification in the sense that its language is *broad* - any signal in θ_1 at some time satisfies it - so the directed distances from other formulae to ϕ_3 are zero. Another notable example is the relation between ϕ_1 and ϕ_4. The directed PH distance is zero in both directions - the two formulae are equivalent. This is due to the fact that any signal that satisfies ϕ_4, already satisfies ϕ_1. The other direction also trivially holds. Some pairs, like ϕ_2 and ϕ_3, have non-zero PH distances in both directions. The SD distances are also shown in upper half of Table I. Here, it can be seen that in most cases, the SD distance is either a lot larger or a lot smaller than the PH distance. This is largely due to the fact that this metric is based on area which is particularly highlighted when comparing any of the formulae to $\top$. Since each formula's satisfaction space is very small in comparison to the entire bounded signal space, each of these values is quite large. In contrast, the SD distance between ϕ_1 and ϕ_5 is fairly small since the satisfaction regions for each of these formulae cover a similar area. The SD distance between ϕ_2 and ϕ_3 is on the larger side as their areas of satisfaction are quite different; however, these areas are still much closer to each other than they are to the entire bounded satisfaction area represented by $\top$. Similar to the PH distance, the SD distance between ϕ_1 and ϕ_4 is zero as they have completely overlapping areas of satisfaction.

The values in Table I illustrate that there are different situations when the PH distance might be favored over the SD distance and vice versa. In cases where one cares about the area covered by the satisfaction region of a formula, the SD distance should be used. For instance, the SD could be used to find a formula close to one that requires a signal to be held at a particular value for a long time interval. However, if one only cares about how close the signal bounds of the formulae are to each other, the PH distance should be used.

TABLE I: Example 2: PH and SD Distances

d_{PH} \ d_{SD}	$\top$	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5	ϕ_6
$\top$	0	0.8	0.76	0.99	0.8	0.804	0.84
ϕ_1	0/0.6	0	0.04	0.19	0	0.036	0.03
ϕ_2	0/0.56	0.04/0	0	0.23	0.04	0.044	0.07
ϕ_3	0/0.6	0.6/0	0.56/0.04	0	0.19	0.186	0.16
ϕ_4	0/0.6	0/0	0/0.04	0/0.6	0	0.036	0.03
ϕ_5	0/0.6	0.6/0	0.56/0.04	0/0.6	0.6/0	0	0.066
ϕ_6	0/0.6	0.6/0	0.56/0.04	0/0.6	0.6/0	0.6/0.04	0

IV. Computation

This section presents algorithms for computing the PH and the SD distances between STL specifications.

A. Pompeiu-Hausdorff Distance

In this section, we propose an optimization-based method to compute the PH distance between two STL formulae.

Definition 4: Given an STL formula φ that contains no negation, we define $\varphi^{\epsilon+}$ with the same logical structure as φ with predicates replaced as follows:

- $f(x) \geq \mu$ replaced with $f(x) \geq \mu - \epsilon$;
- $f(x) \leq \mu$ replaced with $f(x) \leq \mu + \epsilon$.

Intuitively, $\varphi^{\epsilon+}$ is a relaxed version of φ. It is easy to verify from (1) that $\rho(s, \varphi^{\epsilon+}, 0) = \rho(s, \varphi, 0) + \epsilon, \forall s \in \mathcal{S}_T$.

Lemma 1: The following relation holds:

$$\vec{d}_{PH} = \min\{\epsilon \geq 0 \mid \mathcal{L}(\varphi_1) \subseteq \mathcal{L}(\varphi_2^{\epsilon+})\} \tag{8}$$

The following statement provides the main result, and the base for the computational method of this section.

Theorem 2: Given $\varphi_1, \varphi_2 \in \Phi^{\mathcal{S}_T}$, $\mathcal{L}(\varphi_1) \neq \emptyset$, define ϵ^* as the following optimum:

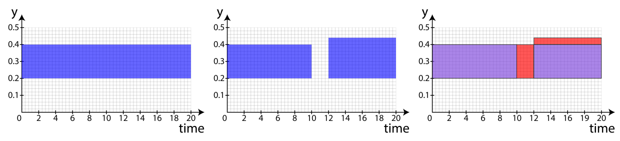
$$\begin{aligned} \epsilon^* = \quad & \max \quad \epsilon, \\ & \text{subject to} \quad s \models \varphi_1, s \not\models \varphi_2^{\epsilon+}, \epsilon \geq 0, s \in \mathcal{S}_T. \end{aligned} \tag{9}$$

Then the following holds:

$$\vec{d}_{PH}(\varphi_1, \varphi_2) = \begin{cases} \epsilon^* & \text{(9) is feasible,} \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

We convert (9) into a MILP problem. The procedure for converting STL into MILP constraints is straightforward, see, e.g., [16]. The encoding details are omitted here. By solving two MILPs, we are able to obtain the PH distance. Two MILPs can be aggregated into a single MILP, but that usually more than doubles the computation time due to larger branch and bound trees. Moreover, it is often useful to have the knowledge of the directed PH distances.

Theorem 2 requires that formulae do not contain negation. Negation elimination is straightforward: first, the formula is brought into its Negation Normal Form (NNF), where all negations appear before the predicates. Next, the predicates are *negated*. For example, we replace $\neg(x \leq \mu)$ by $(x \geq \mu)$. We remind the reader that we do not consider strict inequalities, hence $\neg(x \leq \mu)$ and $(x \leq \mu)$ are both true if $x_0 = \mu$. Finally, observe that the choice of T does not effect the values of PH distance, as long as it is larger than the horizons of two formulae that are compared. Given $\phi_1, \phi_2 \in \Phi^{\mathcal{S}_T}$, the values of s_t for $t > \max\{\|\phi_1\|, \|\phi_2\|\}$ do not have any associated constraints in (9).



(a) ϕ_1 (b) ϕ_5 (c) Overlap ϕ_1 and ϕ_5

Fig. 1: (a) and (b) show the area of satisfaction boxes for ϕ_1 and ϕ_5 from Example 2, respectively. The blue regions represent the boxes that are computed for globally ($\Box$) operators. In (c), the red regions represent the non-overlapping area and the purple regions represent the overlapping area between ϕ_1 and ϕ_5. The SD distance for this example is the area of the red regions $((2\times0.2)+(8\times0.04) = 0.72)$ divided by the maximum time horizon which is $\frac{0.72}{20} = 0.036$.

Complexity: The complexity of (9) is exponential in the number of integers, which grows with the number of predicates and horizons of the formulae. However, since signal values do not have any dynamical constraints, we found solving (9) to be orders of magnitudes faster than comparable STL control problems, such as those studied in [16]. All the values obtained in Table I were evaluated almost instantaneously using Gurobi MILP solver on a personal computer.

B. Symmetric Difference

This section presents an algorithm for computing boxes representing the area of satisfaction of a formula as well as a method for determining the SD between two sets of boxes. Each set of boxes approximates the projection ($\mathcal{P}$) of the formula and represents the valid value-space that a time-varying signal can take such that traces that are contained entirely within the boxes satisfy the formula.

Computing the set of boxes representing the area of satisfaction is a recursive process that takes as input an STL formula, ϕ, a set of max values, X_{max}, for each signal, $x \in X$ (used to normalize the signal values to a unit space), and a discretization threshold, δ. This algorithm, AoS, is presented in Algorithm 1. Here, $box(t_1, t_2, x_1, x_2, i) = \mathcal{P}(\Box_{[t_1,t_2]} x_1 \leq x^i \leq x_2) \subseteq \mathbb{S} \times T\mathbb{U}$ creates a new box with minimum and maximum times t_1 and t_2, minimum and maximum values x_1 and x_2, and spatial dimension $i \in \{1, \ldots, n\}$, respectively; $overlap$ determines the time window of the overlap between two boxes; $combine$ takes two boxes and produces a set of boxes representing the intersection of the overlapping time window region; $b.lt$, $b.ut$, $b.lv \in \mathbb{S}$, and $b.uv \in \mathbb{S}$ return the lower time window, upper time window, lower variable, and upper variable values for box b, respectively; $*$ in the definition of a box denotes that it may restrict multiple spatial dimensions; and the $\|$ operator is used to create a "choice" set representing that either of the two sets can be selected as the set of boxes representing the area of satisfaction.

To address the problem of projection of formulae containing disjunction (the converse to Theorem 1), AoS utilizes the $\|$ operator. If this algorithm instead generated boxes representing the projection of all formulae, it would be possible for the satisfaction space represented by the boxes to capture signals that the original formula does not allow.

ALGORITHM 1: Convert to Area of Satisfaction Boxes (AoS)

```
input : STL formula $\phi$, max value set $X_{max}$, discretization threshold $\delta$.
output: Set of boxes $\mathcal{B}$ for each signal in $\phi$.
if $\phi := x^i \leq \pi$ then return $box(0,0,0,\pi/x_{max},i)$
else if $\phi := x^i > \pi$ then return $box(0,0,\pi/x_{max},1,i)$
else if $\phi := \phi_1 \wedge \phi_2$ then
    Create a new set $\mathcal{B}$
    for each $b_1 \in AoS(\phi_1)$ and each $b_2 \in AoS(\phi_2)$ do
        if $overlap(b_1,b_2)$ then Add $combine(b_1,b_2)$ to $\mathcal{B}$
        else Add $b_1$ to $\mathcal{B}$ and $b_2$ to $\mathcal{B}$
    return $\mathcal{B}$
else if $\phi := \phi_1 \vee \phi_2$ then return $AoS(\phi_1) \parallel AoS(\phi_2)$
else if $\phi := \square_{[t_1,t_2]}(\phi_1)$ then
    Create a new set $\mathcal{B}$
    for each $b \in AoS(\phi_1)$ do
        $b' = box(b.lt + t_1, b.ut + t_2, b.lv, b.uv, *)$
        Add $b'$ to $\mathcal{B}$
    return $\mathcal{B}$
else if $\phi := \diamondsuit_{[t_1,t_2]}(\phi_1)$ then
    return $AoS\left(\bigvee_{i=1}^{(t_2-t_1)/\delta} \square_{[t_1+\delta(i-1),t_1+\delta i]}(\phi_1)\right)$
```

The application in Section V highlights this problem and presents a way of dealing with it for that particular example.

For operators such as globally ($\square$), AoS is exact and produces boxes bound by the time bounds of the operator that represent the projection of the primitive. However, operators such as eventually ($\diamondsuit$) do not immediately lend themselves to conversion into a set of boxes. In order to deal with this operator, we approximate it by converting it into a disjunction of globally predicates. Each globally predicate is generated using a small threshold value (δ) for its time window width. The new formula requires that the expression be true in at least one of the smaller time windows essentially introducing a mandatory δ "hold" time for eventually operators. The tunability of δ allows for a user to give up some accuracy for gains in performance of box computation and ultimately distance comparison. Examples of computing the area of satisfaction boxes in Example 2 are shown in Figure 1.

The SD between two sets of boxes is computed by calculating the area of the sum of the non-intersected area for each box set. This value is normalized by the maximum time horizon, T, and results in the SD computation:

$$d_{SD}(\mathcal{B}_{\phi_1},\mathcal{B}_{\phi_2}) = \frac{1}{T+1}\left|\left(\bigcup_{b_1\in\mathcal{B}_{\phi_1}} b_1\right)\triangle\left(\bigcup_{b_2\in\mathcal{B}_{\phi_2}} b_2\right)\right|$$

Figure 1c illustrates how the SD between ϕ_1 and ϕ_5 is computed. The SD distance is scaled by $\frac{T+1}{T'+T+1}$ if the maximum horizon is increased by T'. This again shows the temporal nature of the SD as opposed to the PH distance which does not change.

Complexity: The complexity of Algorithm 1 depends on the complexity of the $\parallel$ operation, which may be exponential depending on how it is implemented. Otherwise, the algorithm is polynomial due to the box combination operations carried out whenever a conjunction predicate is encountered. In practice, running Algorithm 1 on formulae with a few dozen predicates typically takes only a few seconds.

V. QUANTIFICATION OF DESIGN QUALITY

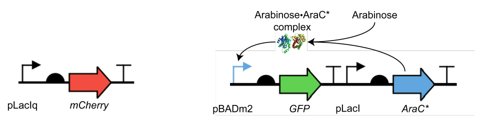
In this application, we show an example of how the proposed metrics can be used in behavioral synthesis. Behavioral synthesis is an important process in design automation where the description of a desired behavior is interpreted and a system is created that implements the desired behavior. Our goal is to check if the characterized implementations satisfy the specifications of a system. Implementations include simulations and execution traces of a system. These implementations are characterized into formal specifications using TLI. We show that the proposed metrics can be used in the synthesis step to choose a design from the solution space that can best implement the desired specification. The specific example we have chosen to highlight this application is the synthesis of genetic circuits in synthetic biology.

In this example, we have a set of desired behaviors in STL which describe the various behaviors expected of a genetic circuit. This set of behaviors is referred to as a performance specification: $\mathcal{S}^{\phi}$. $\mathcal{S}^{\phi}$ consists of 2 STL formulae: ϕ_{low} and ϕ_{high} which describe the desired amount of output produced by the genetic circuit over time:

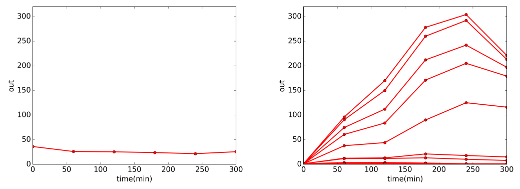
$$\begin{aligned}\phi_{low} &= \square_{[0,300]}(x<40) \wedge \square_{[0,300]}(x>0)\\ \phi_{high} &= \square_{[0,125]}(x<200) \wedge \square_{[125,300]}(x<320) \wedge\\ &\quad \square_{[0,150]}(x>0) \wedge \square_{[150,200]}(x>100) \wedge\\ &\quad \square_{[200,300]}(x>150)\end{aligned}$$

In this case, the output of the circuit corresponds to the expression of a fluorescent protein. ϕ_{low} specifies that the output must consistently be below 40 units from time 0 to 300, and ϕ_{high} specifies that that output must gradually increase over time and must end up between 150 and 320 units between time 200 and 300. Our solution space consists of two genetic circuits. The first circuit has a constitutive promoter as shown in Figure 2a. Constitutive expression removes flexibility for consistency allowing constant protein production independent of the state or inputs of the system, which is highlighted in Figure 2c. The second circuit has an inducible promoter: a sugar detecting transcription factor AraC*, which will turn on the protein production if and only if it is in the presence of a specific input molecule (arabinose) as shown in Figure 2b. Figure 2d shows the output of the circuit for various concentrations of arabinose. Both of these synthetic genetic circuits were built in *Escherichia coli*. The traces were obtained from biological experiments by measuring fluorescence.

Our goal is to choose the circuit that can "satisfy as many behaviors as possible" in $\mathcal{S}^{\phi}$. Note that it is difficult to express the term "satisfy as many behaviors as possible" using the syntax and semantics of STL. For instance, expressing the desired specification as a disjunction of all the formulae in $\mathcal{S}^{\phi}$ would imply that satisfying any one specification is sufficient for the genetic circuit to satisfy the performance specification. Similarly, expressing the desired specification as a conjunction of all the formulae in $\mathcal{S}^{\phi}$ would imply that at any point in time, the output of a genetic circuit must have multiple distinct values, which is physically impossible.

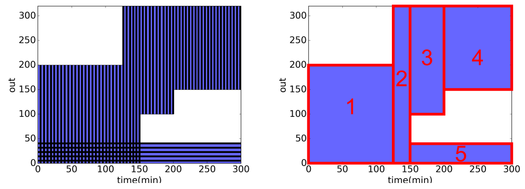


(a) Constitutive Expression (b) Induction Circuit



(c) Constitutive Expression (d) Induction Circuit

Fig. 2: (a) and (b) show SBOL visual representations of the genetic circuits with a constitutive promoter and an inducible promoter, respectively. Traces in (c) and (d) were obtained by evaluating geometric mean fluorescence at regular intervals by flow cytometry.



(a) $\mathcal{P}(\phi_{low} \vee \phi_{high})$ (b) $\mathcal{B}_{\phi_{low}} \cup \mathcal{B}_{\phi_{high}} = \mathcal{B}_{\mathcal{S}^\phi}$

Fig. 3: (a) the union of the areas of satisfaction for ϕ_{low} and ϕ_{high}. The vertical and horizontal stripe areas represent $\mathcal{P}(\phi_{low})$ and $\mathcal{P}(\phi_{high})$, respectively. (b) the boxes created for $\mathcal{B}_{\phi_{low}} \cup \mathcal{B}_{\phi_{high}}$.

This conundrum is highlighted in the current example. The output of constitutive expression satisfies ϕ_{low} but cannot satisfy ϕ_{high}. The induction circuit produces traces that can satisfy both ϕ_{low} and ϕ_{high}. However, traditional model checking techniques may not help a designer choose the desired circuit. Using statistical model checking, for example, the circuit with constitutive expression yields a satisfaction likelihood of 1.0 and the induction circuit yields a satisfaction likelihood of 0.83 against $\phi_{low} \vee \phi_{high}$. With these results, one might think that the circuit with constitutive expression best satisfies the performance specification.

To address the issue of satisfying as many behaviors as possible, we treat the performance specification's region of satisfaction as the union of the regions of satisfaction of all the formulae in $\mathcal{S}^\phi$ as shown in Figure 3a. We compute this region by taking the union of the generated boxes for each formula that are computed using Algorithm 1. The union of the box sets of all STL formulae in $\mathcal{S}^\phi$ is represented as $\mathcal{B}_{\phi_{low}} \cup \mathcal{B}_{\phi_{high}} = \mathcal{B}_{\mathcal{S}^\phi}$ and is shown in Figure 3b. Using Grid TLI [17], we produce STL formulae, ϕ_{con} and ϕ_{ind}, for each circuit using the traces shown in Figures 2c and 2d, respectively. We then use the SD metric and get the following values: $d_{SD}(\mathcal{B}_{\mathcal{S}^\phi}, \mathcal{B}_{\phi_{con}}) = 0.636$ and $d_{SD}(\mathcal{B}_{\mathcal{S}^\phi}, \mathcal{B}_{\phi_{ind}}) = 0.304$. Using the PH metric, we get: $d_{PH}(\mathcal{B}_{\mathcal{S}^\phi}, \mathcal{B}_{\phi_{con}}) = 0.067$ and $d_{PH}(\mathcal{B}_{\mathcal{S}^\phi}, \mathcal{B}_{\phi_{ind}}) = 0$. These results imply that the behavior of the induction circuit is closer to the desired specification than the circuit with constitutive expression, and thus, it should be selected as the desired circuit.

VI. Future Work

An immediate theoretical extension is studying continuous-time signals. By assuming Lipschitz continuity of signals, it is possible to provide bounds between the metrics computed in discrete-time and the ones in continuous-time. A similar idea was used in [6] to compute sampled-time STL scores. The second extension is relaxing the assumption on rectangular predicates.

References

[1] C. Baier and J. Katoen, *Principles of model checking*. MIT Press, 2008.

[2] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE transactions on robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[3] G. Batt, B. Yordanov, R. Weiss, and C. Belta, "Robustness analysis and tuning of synthetic gene networks," *Bioinformatics*, vol. 23, no. 18, pp. 2415–2422, 2007.

[4] S. Coogan, M. Arcak, and C. Belta, "Formal methods for control of traffic flow: Automated control synthesis from finite-state transition models," *IEEE Control Systems*, vol. 37, no. 2, pp. 109–128, 2017.

[5] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 2004, pp. 152–166.

[6] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, vol. 410, no. 42, pp. 4262–4291, 2009.

[7] A. Donzé, T. Ferrere, and O. Maler, "Efficient robust monitoring for STL," in *Computer Aided Verification*. Springer, 2013, pp. 264–279.

[8] C.-I. Vasile, J. Tumova, S. Karaman, C. Belta, and D. Rus, "Minimum-violation scLTL motion planning for mobility-on-demand," in *IEEE Intl Conference on Robotics and Automation*, 2017, pp. 1481–1488.

[9] C. I. Vasile, D. Aksaray, and C. Belta, "Time Window Temporal Logic," *Theoretical Computer Science*, vol. 691, no. Supplement C, pp. 27–54, August 2017.

[10] K. Kim, G. Fainekos, and S. Sankaranarayanan, "On the minimal revision problem of specification automata," *The International Journal of Robotics Research*, 2015.

[11] S. Ghosh, D. Sadigh, P. Nuzzo, V. Raman, A. Donzé, A. L. Sangiovanni-Vincentelli, S. S. Sastry, and S. A. Seshia, "Diagnosis and repair for synthesis from signal temporal logic specifications," in *Hybrid Systems: Computation and Control*, 2016, pp. 31–40.

[12] J. R. Munkres, *Topology*. Prentice Hall, 2000.

[13] J. B. Conway, *A course in abstract analysis*. American Mathematical Soc., 2012, vol. 141.

[14] A. Dokhanchi, B. Hoxha, and G. Fainekos, *5th International Conference on Runtime Verification, Toronto, ON, Canada*. Springer, 2014, ch. On-Line Monitoring for Temporal Logic Robustness, pp. 231–246.

[15] D. Kraft, "Computing the Hausdorff distance of two sets from their signed distance functions," *Computational Geometry*, Submitted March 2015.

[16] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *Conference on Decision and Control (CDC)*. IEEE, 2014, pp. 81–87.

[17] P. Vaidyanathan, R. Ivison, G. Bombara, N. A. DeLateur, R. Weiss, D. Densmore, and C. Belta, "Grid-based temporal logic inference," in *IEEE Conference on Decision and Control*, 2017, pp. 5354–5359.