A User Guide for TaLiRo

Georgios E. Fainekos[1] and George J. Pappas[1,2]

[1] Department of Computer and Information Science, Univ. of Pennsylvania
`fainekos (at) cis.upenn.edu`
[2] Department of Electrical and Systems Engineering, Univ. of Pennsylvania
`pappasg (at) ee.upenn.edu`

Abstract. TaLiRo (TemporAl LogIc RObustness) is a tool for the computation of the robustness of a propositional temporal logic specification with respect to a discrete time signal. This document provides a brief introduction to Linear and Metric Temporal Logics, describes the usage of the toolbox and concludes with several examples. This guide refers to version v0.1 of TaLiRo.

1 Introduction

Model checking [1] has been proven to be a very useful tool for the verification of software and hardware systems. The tools and methodologies developed for such systems do not naturally extend to systems whose state space is some general infinite space, for example continuous and hybrid systems. In this case, the model checking problem becomes harder and in most of the cases is undecidable [2]. In practice, the validation of such systems still relies heavily on methods that involve systematic testing [3, 4]. More recently, temporal logic testing [5, 6] has been proposed as a framework that can provide us with additional information about the properties of continuous or discrete time signals.

However, the classical approaches to temporal logic testing involve a Boolean abstraction of the value of the signal with respect to the atomic propositions in the formula. This loss of information can be quite critical when we consider systems that model or control physical processes. For example, consider the signals s_1 and s_2 in Fig. 1. Both of them satisfy the same specification "if the value of the signal drops below -10, then it should also raise above 10 within 2 time units". Nevertheless, a visual inspection of Fig. 1 indicates that there exists a qualitative difference between s_1 and s_2. The later "barely" satisfies the specification. Indeed as we can see in Fig. 2, adding a bounded noise on s_2 renders the property unsatisfiable on s_2.

In order to differentiate between such trajectories of a system, we introduced the concept of *robustness estimate* in [7]. Informally, the robustness estimate is a bound on the perturbation that the signal can tolerate without changing the truth value of a specification expressed in Linear [8] or Metric Temporal Logic [9]. For example, signal s_1 in Fig. 1 has robustness estimate 5 and signal s_2 has robustness estimate 0.5.

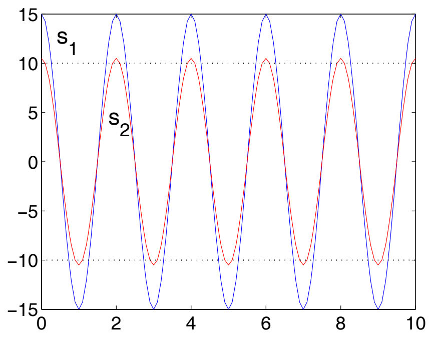

Fig. 1. Two signals s_1 and s_2 which satisfy the specification: $\Box(p_1 \rightarrow \Diamond_{\leq 2} p_2)$. Here, $\mathcal{O}(p_1) = \mathbb{R}_{\leq -10}$ and $\mathcal{O}(p_2) = \mathbb{R}_{\geq 10}$.

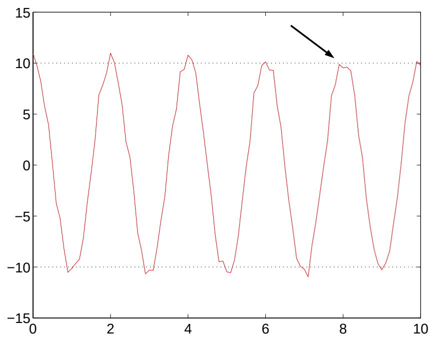

Fig. 2. The signal s_2 modified by random noise. The arrow points to the point in time where the property fails.

This document is the user guide for TaLiRo. TaLiRo computes the robustness estimate [7] of a temporal logic specification (LTL or MTL) over a finite timed state sequence, that is, a bounded duration discrete time signal marked with timestamps. Determining the robustness estimate of a signal is useful in cases where the system has known bounded measurement noise and bias. The robustness estimate has also been proven useful in temporal logic verification of dynamical systems [10] and in continuous time reasoning using discrete time methods [11].

2 Theoretical Background

In this section, we provide a brief overview of timed state sequences and temporal logics. Also, we give the intuition behind the robustness estimate.

Let $\mathbb{R}$ be the set of the real numbers, $\mathbb{Q}$ be the set of rational numbers and $\mathbb{N}$ the set of the natural numbers. We denote the extended real number line by $\overline{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$. In addition, we use pseudo-arithmetic expressions to represent certain subsets of the aforementioned sets. For example, $\mathbb{R}_{\geq 0}$ denotes the subset of the reals whose elements are greater than or equal to zero. We let $\mathbb{B} = \{\bot, \top\}$, where $\top$ and $\bot$ are the symbols for the boolean constants *true* and *false* respectively. Finally, given a set A, $\mathcal{P}(A)$ denotes the powerset of the set A and $|A|$ denotes its cardinality.

2.1 Signals and Timed State Sequences

In nature, physical processes evolve in a continuous manner. Think for example, the temperature in a room or the height of the water in a tank. This does not imply that the value of the property that we measure evolves in a continuous manner (e.g., the stress in a rod that is going to break), but rather that there exists a value at each point in time. The most natural way to model the evolution of such quantities in time is through functions defined over the reals.

In that sense, a *continuous time signal* s is a simply a function $s : R \to \mathbb{R}^n$ from a time domain R to some value in the set $\mathbb{R}^n$ with $n \in \mathbb{N}$. When we consider bounded time signals, then we let $R = [0, r] \subseteq \mathbb{R}_{\geq 0}$, otherwise we let $R = \mathbb{R}_{\geq 0}$. As an example of a continuous time signal, consider the function $s_1(t) = \sin t + \sin 2t$ in Fig. 3 such that $R = [0, 7\pi]$. Note that the range of a signal is $\mathbb{R}^n$ rather than just $\mathbb{R}$ since in many cases we want to model several physical quantities using only one variable. For example, we might be taking measurements from 5 temperature sensors in a room.

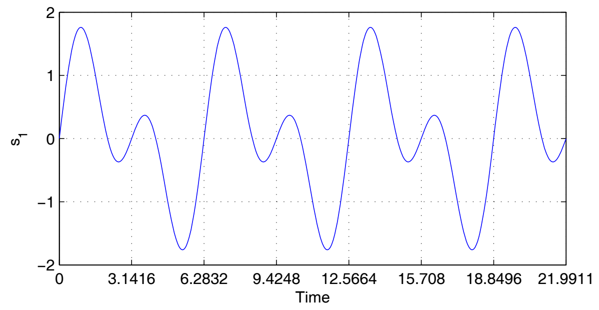

Fig. 3. A continuous time signal s_1 with time domain $R = [0, 7\pi]$.

However, in virtually all the practical cases the representation of the behavior of such systems that is available to us for analysis is in discrete time. For example when we monitor the temperature in a room, we cannot know the value of the continuous time signal at all points in time, but only at those points in time that are attainable through sampling. This is also true, when we simulate a continuous time signal using a digital computer. Some form of discretization of time is always necessary.

A *discrete time signal* σ can represent computer simulated trajectories of physical models or the sampling process that takes place when we digitally monitor physical systems. Informally, a discrete time signal σ is a sequence of snapshots of the continuous time behaviour of a system. Each such snapshot represents the state of the system at a particular point in time (see Fig. 4). Nevertheless, a discrete time signal does not provide us with any timing information. In order to reason about the timing properties of a discrete time signal, we introduce the *timing function* τ. The role of the timing function is to pair each snapshot of σ with a time stamp.

More formally, we define a discrete time signal σ to be a function from N to $\mathbb{R}^n$. Such a signal can be of bounded or unbounded duration. In the former case we set $N = \mathbb{N}_{\leq m}$ for some $m \in \mathbb{N}$, while in the latter $N = \mathbb{N}$. Analogously, τ is a function $\tau : N \to \mathbb{R}_{\geq 0}$. Two important restrictions on a timing function τ are

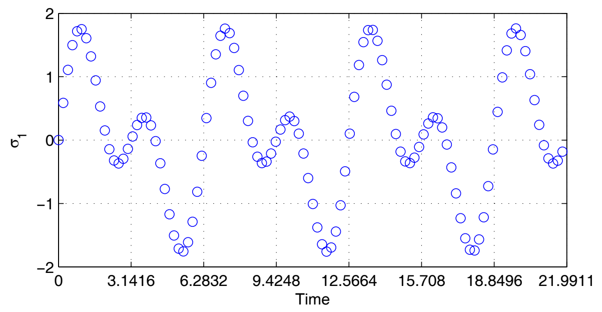

Fig. 4. A discrete time signal $\sigma_1(i) = \sin\tau_1(i) + \sin 2\tau_1(i)$ where the timing function is $\tau_1(i) = 0.2i$.

1. τ must be a strictly increasing function, i.e., $\tau(i) < \tau(j)$ for $i < j$.
2. if N is infinite, then τ must diverge, i.e., $\lim_{i\to+\infty} \tau(i) = +\infty$.

In order to reason about the properties of signals, we have to define certain sets of interest in $\mathbb{R}^n$. For example in Fig. 1, we would like to know whether the value of the signal is in the sets $(-\infty, -10]$ and $[10, +\infty)$. We do so by using a set of atomic propositions AP which label subsets of $\mathbb{R}^n$. In other words, we define an observation map $\mathcal{O} : AP \to \mathcal{P}(\mathbb{R}^n)$ such that for each $p \in AP$ the corresponding set is $\mathcal{O}(p) \subseteq \mathbb{R}^n$. Also, we define the "inverse" map $\mathcal{O}^{-1} : \mathbb{R}^n \to \mathcal{P}(AP)$ as $\mathcal{O}^{-1}(x) := \{p \in AP \mid x \in \mathcal{O}(p)\}$ for $x \in \mathbb{R}^n$.

By pairing a sequence of atomic propositions $\bar{\sigma} = \mathcal{O}^{-1} \circ \sigma$ with a timing function τ, we define what is usually referred to as a *timed state sequence* $\mu = (\bar{\sigma}, \tau)$. Here, $\circ$ denotes function composition : $(f \circ g)(t) = f(g(t))$. Timed state sequences are a widely accepted model for reasoning about real time systems [12]. In the following, we let $\mu^{(1)}$ be the first member of the pair, i.e., $\mu^{(1)} = \mathcal{O}^{-1} \circ \sigma$, and $\mu^{(2)}$ be the second member of the pair, i.e., $\mu^{(2)} = \tau$. Finally, $|\mu|$ denotes the length of the timed state sequence. That is, if the time domain of σ and τ is N, then $|\mu| = \sup N + 1$.

2.2 Metric Temporal Logic

The Metric Temporal Logic (MTL) was introduced in [9] in order to reason about the quantitative timing properties of boolean signals. In this section, we review the basics of propositional MTL.

In order to make apparent the use of MTL for signal analysis purposes, we first give an informal description of the operators of the logic. MTL formulas are built over a set of propositions, the set AP in our case, using combinations of the traditional and temporal operators. Traditional logic operators are the *conjunction* ($\wedge$), *disjunction* ($\vee$), *negation* ($\neg$), *implication* ($\rightarrow$) and *equivalence* ($\leftrightarrow$). Some of the temporal operators, which we will be using here, are *next* ($\bigcirc_{\mathcal{I}}$), *weak next* ($\circledsim_{\mathcal{I}}$), *eventually* ($\Diamond_{\mathcal{I}}$), *always* ($\Box_{\mathcal{I}}$), *until* ($\mathcal{U}_{\mathcal{I}}$) and *release* ($\mathcal{R}_{\mathcal{I}}$). The subscript $\mathcal{I}$ imposes timing constraints on the temporal operators. The interval

$\mathcal{I}$ can be open, half-open or closed, bounded or unbounded, but it must be non-empty ($\mathcal{I} \neq \emptyset$). Metric Temporal Logic can describe the usual properties of interest for control design problems:

- **(Bounded time) Reachability:** The specification $\Diamond_{[1,5)} p_1$ where $\mathcal{O}(p_1) = (-\infty, -10]$ requires that the value of signal should be in the set $(-\infty, -10]$ within 1 to 5 time units (excluding 5).
- **Safety:** The formula $\Box_{[0,+\infty)} \neg p_1$ requires that for all time $i \geq 0$ (0 is the current point in time) $\sigma(i)$ should not be in the set $\mathcal{O}(p)$, i.e., $\sigma(i) \notin \mathcal{O}(p)$.

Beyond the usual properties, MTL can capture sequences of events and infinite behaviours. For example:

- **Reachability while avoiding regions:** The formula $\neg(p_1 \vee p_2 \vee p_3) \mathcal{U}_{\mathcal{I}} p_4$ expresses the property that the sets $\mathcal{O}(p_i)$ for $i = 1, 2, 3$ should be avoided until $\mathcal{O}(p_4)$ is reached at a time within $\mathcal{I}$.
- **Sequencing:** The requirement that the signal must take values in the sets $\mathcal{O}(p_1)$, $\mathcal{O}(p_2)$ and $\mathcal{O}(p_3)$ in that order is captured by the formula $\Diamond_{\mathcal{I}_1}(p_1 \wedge \Diamond_{\mathcal{I}_2}(p_2 \wedge \Diamond_{\mathcal{I}_3} p_3))$. Note, moreover, that $\sigma(i)$ should be in $\mathcal{O}(p_3)$ at some time in $\mathcal{I}_1 \oplus \mathcal{I}_2 \oplus \mathcal{I}_3$, where $\oplus$ is the Minkowski sum.
- **Coverage:** Formula $\Diamond p_1 \wedge \Diamond p_2 \wedge \Diamond p_3$ reads as the signal must eventually take a value in the set $\mathcal{O}(p_1)$ and it must eventually take a value in the set $\mathcal{O}(p_2)$ and it must eventually take a value in the set $\mathcal{O}(p_3)$. Thus, we require the signal to eventually take values from all the sets of interest without imposing any ordering or timing constraints.
- **Recurrence:** The formula $\Box_{[0,10]}(\Diamond p_1 \wedge \Diamond p_2 \wedge \Diamond p_3)$ requires that the signal does whatever the coverage specification does and, in addition, will force the system to repeat the desired objective for up to 10 time units. Note that $\Box_{[0,10]}(\Diamond p_1 \wedge \Diamond p_2 \wedge \Diamond p_3)$ will be true even if the subformula $\Diamond p_1 \wedge \Diamond p_2 \wedge \Diamond p_3$ becomes true only once, specifically, after the 10 time units have passed.
- **Sampling:** If we would like to find out whether the next sample will be taken within 0.9 and 1.1 time units and whether it remains within the domain of interest $\mathcal{O}(p)$, then we can write the formula $\bigcirc_{(0.9,1.1)} p$.

More complicated specifications can be composed from the basic specifications using the logic operators. An excellent applied introduction to LTL for software and hardware verification can be found in [13]. Now, we briefly present some formal definitions.

Definition 1 (MTL Syntax). *Let AP be the set of atomic propositions and $\mathcal{I}$ be any non-empty interval of $\mathbb{R}_{\geq 0}$. The set $\Phi_{\mathbb{B}}$ of all well-formed formulas (wff) is inductively defined using the following grammar:*

$$\phi \ ::= \ b \mid p \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \ \bigcirc_{\mathcal{I}} \phi \mid \circleddash_{\mathcal{I}} \phi \mid \phi \mathcal{U}_{\mathcal{I}} \phi \mid \phi \mathcal{R}_{\mathcal{I}} \phi$$

where $b \in \mathbb{B}$ and $p \in AP$.

Sometimes for clarity in the presentation, we replace $\mathcal{I}$ with pseudometric expressions, e.g., $\mathcal{U}_{[0,1]}$ is written as $\mathcal{U}_{\leq 1}$. In the case where $\mathcal{I} = [0, +\infty)$, we remove the subscript $\mathcal{I}$ from the temporal operators, i.e., we just write $\mathcal{U}$ and $\mathcal{R}$. When all the subscripts of the temporal operators are of the form $[0, +\infty)$, then the MTL formula ϕ reduces to an LTL formula [8, 14]. Comparing the two logics, MTL is used to reason about the quantitative timing properties, whereas LTL only about qualitative timing properties.

Metric Temporal Logic (MTL) formulas are interpreted over timed state sequences. In this paper, we define MTL semantics using the relation $\models$. When we write $(\mu, i) \models \phi$, we mean that the timed state sequence μ satisfies formula ϕ at time i. If μ does not satisfy formula ϕ at time i, then we write $(\mu, i) \not\models \phi$. In the definition below, we also use the following notation :

– For $P \subseteq \mathbb{R}_{\geq 0}$, we define $\tau^{-1}(P) := \{i \in \mathbb{N} \mid \tau(i) \in P\}$.
– For $\mathcal{I} \subseteq \mathbb{R}_{\geq 0}$ and for any $t \in \mathbb{R}_{\geq 0}$, we define $t + \mathcal{I} := \{t + t' \mid t' \in \mathcal{I}\}$.

Definition 2 (MTL Semantics). *Let $\phi \in \Phi_{\mathbb{B}}$ be an MTL formula and μ be a timed state sequence, then the semantics of ϕ is defined by*

$$
\begin{aligned}
&(\mu, i) \models \top \\
&(\mu, i) \not\models \bot \\
&(\mu, i) \models p \text{ iff } p \in \bar{\sigma}(i) \\
&(\mu, i) \models \neg\phi_1 \text{ iff } (\mu, i) \not\models \phi_1 \\
&(\mu, i) \models \bigcirc_{\mathcal{I}}\phi_1 \text{ iff } \begin{cases} \tau(i+1) \in (\tau(i) + \mathcal{I}) \wedge (\mu, i+1) \models \phi_1 & \text{if } i < |\mu| - 1 \\ \bot & \text{otherwise} \end{cases} \\
&(\mu, i) \models \circleddash_{\mathcal{I}}\phi_1 \text{ iff } \begin{cases} \tau(i+1) \in (\tau(i) + \mathcal{I}) \to (\mu, i+1) \models \phi_1 & \text{if } i < |\mu| - 1 \\ \top & \text{otherwise} \end{cases} \\
&(\mu, i) \models \phi_1 \vee \phi_2 \text{ iff } (\mu, i) \models \phi_1 \vee (\mu, i) \models \phi_2 \\
&(\mu, i) \models \phi_1 \wedge \phi_2 \text{ iff } (\mu, i) \models \phi_1 \wedge (\mu, i) \models \phi_2 \\
&(\mu, i) \models \phi_1 \mathcal{U}_{\mathcal{I}} \phi_2 \text{ iff } \exists j \in \tau^{-1}(\tau(i) + \mathcal{I}) \cap N \,.\, (\mu, j) \models \phi_2 \wedge \forall k \in [i, j) \,.\, (\mu, k) \models \phi_1 \\
&(\mu, i) \models \phi_1 \mathcal{R}_{\mathcal{I}} \phi_2 \text{ iff } \forall j \in \tau^{-1}(\tau(i) + \mathcal{I}) \cap N \,.\, (\mu, j) \models \phi_2 \vee \exists k \in [i, j) \,.\, (\mu, k) \models \phi_1
\end{aligned}
$$

where $i, j, k \in \mathbb{N}$, $\bar{\sigma} = \mu^{(1)}$, $\tau = \mu^{(2)}$ and $N = \{0, 1, \ldots, |\mu| - 1\}$.

Informally, the formula $\phi_1 \mathcal{U}_{\mathcal{I}} \phi_2$ expresses the property that over the sequence $\bar{\sigma}$ and in the time interval $\tau^{-1}(\tau(i) + \mathcal{I}) \cap N$, there exists some time that $\bar{\sigma}$ makes ϕ_2 true and, furthermore, for all previous times, $\bar{\sigma}$ satisfies ϕ_1. Intuitively, the release operator $\phi_1 \mathcal{R} \phi_2$ states that ϕ_2 should always hold during the interval $\tau^{-1}(\tau(i) + \mathcal{I}) \cap N$, a requirement which is released when ϕ_1 becomes true. In the above definition, the intersection with the set N has been added because we are using finite timed state sequences. We can also define the derived temporal operators *eventually* $\Diamond_{\mathcal{I}}\phi = \top \mathcal{U}_{\mathcal{I}} \phi$ and *always* $\Box_{\mathcal{I}}\phi = \bot \mathcal{R}_{\mathcal{I}} \phi$ which are self-explanatory.

2.3 Robustness Estimate

We proceed on to give an informal explanation of the robustness estimate [7]. Given a timed state sequence μ and a formula ϕ, the robustness estimate is a number ε from the closure of the reals $\overline{\mathbb{R}}$. The intuitive meaning of ε is that if $\varepsilon > 0$, then any timed state sequence μ' that remains ε-close to μ satisfies the same specification ϕ as μ. On the other hand, if $\varepsilon < 0$, then we know that μ does not satisfy the specification and, moreover, that any other timed state sequence μ' in the $|\varepsilon|$-neighborhood of μ also does not satisfy ϕ. The details of how the robustness estimate can be computed can be found in [15, 16].

From an application perspective, it is important to clearly define what we mean by saying that two timed state sequences remain $|\varepsilon|$-close. For that, we first need the notion of the supremum metric. Given two discrete time signals σ_1 and σ_2, the supremum metric ρ is defined as $\rho(\sigma_1, \sigma_2) = \sup_{i \in N} \|\sigma_1(i) - \sigma_2(i)\|$, where $\|\cdot\|$ is the Euclidean norm and N is the discrete time signals' time domain, i.e., $N = \mathbf{dom}(\sigma_1) = \mathbf{dom}(\sigma_2)$. In other words, the supremum metric returns the maximum distance between the two signals at each point in time. An important assumption here is that the two signals σ_1 and σ_2 are sampled at the same points in time. That is, if τ_1 and τ_2 are the corresponding sampling functions of σ_1 and σ_2, then for all $i \in N$ we have $\tau_1(i) = \tau_2(i)$. If we let $\mu_1 = (\mathcal{O}^{-1} \circ \sigma_1, \tau_1)$ and $\mu_2 = (\mathcal{O}^{-1} \circ \sigma_2, \tau_2)$, then we can extend the definition of ρ over timed state sequences as follows : $\rho(\mu_1, \mu_2) = \rho(\sigma_1, \sigma_2)$ (again under the assumption that the two sampling functions τ_1 and τ_2 are the same).

Example 1. Consider again the discrete time signal σ_1 in Fig. 4 and assume that we are given the LTL specification $\phi_0 = \Box p_1$, where $\mathcal{O}(p_1) = [-2, 2]$. Then, the robustness estimate is $\varepsilon = 0.2403$. Note that any other timed state sequence that remains within the tube around σ_1 in Fig. 5 also satisfies the specification ϕ_0.

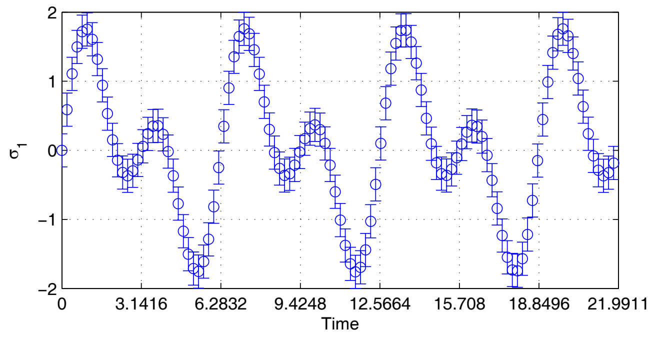

Fig. 5. The signal σ_1. The bars indicate the range of values that any other timed state sequence can take in order to satisfy the specification ϕ_0.

3 TaLiRo

TaLiRo is a tool that computes the robustness estimate ε of an MTL formula ϕ with respect to a finite timed state sequence μ. Version 0.1 is available in two formats (Windows and Linux) and it supports only 1D signals. When TaLiRo is executed without any input arguments, i.e., `taliro`, then it takes as input arguments the demo input files `demo_spec.txt` and `demo_data.txt` that are distributed with the tool. The input arguments to `taliro` must be two input files, e.g., `taliro inputspec.txt inputdata.txt`.

This is a console application. In order to execute TaLiRo, you have to open a console window in the MS Windows family products : `start -> run`, then type `cmd` and change to the directory where you have unzipped the software package. Note that in certain versions of Linux systems you might have to type `./taliro` instead of `taliro` in order to run the software.

3.1 Explanation of Input Arguments

Usage `taliro inputspec.txt inputdata.txt`

The file `inputspec.txt`, as the name implies, includes the MTL or LTL formula as well as the observation map $\mathcal{O}$ and some auxiliary variables. A typical input in ASCII format is the following.

```
% Demo input specification file for TaLiRo
% G. Fainekos - GRASP Lab - 2008.01.22

[](p1-><>_(0,.5) !p1)

signal dimension : 1

number of predicates : 3

p1 number of constraints : 1
-1.0 -1.0

p2 number of constraints : 2
1.0 0.5
-1.0 0.5

p3 number of constraints : 1
1.0 -1.0

timing constraints on the number of samples : no
number of samples : 3142
```

The lines that start with the special character % are comment lines, e.g. lines `01` and `02`. The empty lines, e.g., `03`, `05`, are not required, however they make the text more readable.

Line `04` is the MTL or LTL formula. Table 1 indicates the correspondence between the symbols of the logical operators and the input ASCII characters. The timing constraints on the temporal operators follow the temporal operator

$\neg$	$\vee$	$\wedge$	$\rightarrow$	$\leftrightarrow$	
`!`	`\/`	`/\`	`->`	`<->`	
$\bigcirc$	$\tilde{\bigcirc}$	$\Box$	$\Diamond$	$\mathcal{U}$	$\mathcal{R}$
`X`	`W`	`[]`	`<>`	`U`	`R`

Table 1. Correspondence between logical operators and ASCII symbols.

using an underscore. That is, if `T` $\in$ { `X, W, [], <>, R, U` }, then `T_I` is a temporal operator with timing constraints. In turn, the timing constraints `I` can have the form $\ll a, b \gg$, where $\ll \in$ { `(`, `[` }, $\gg \in$ { `)`, `]` } and $a, b \in \mathbb{Q} \cup \{\pm$`inf`$\}$. Currently, no negative numbers are allowed in the timing constraints `I` since we do not consider past operators. Some examples of timing constraints on the temporal operators are :

`U_(0.23,5.12)`, `[]_[0,30]`, `R_[10,inf)`, `X_(3.2331,inf)`, `<>_[2,2]`

Finally, if there is no timing constraint after a temporal operator, then `I = [0,+inf)` is implied[3].

If the timing constraints refer to the actual evolution of time, that is, to the timestamps $\tau(i)$ of a timed state sequence $\mu = (\bar{\sigma}, \tau)$, then we have to store the bounds of `I` using double precision floating point variables. In this case, comparisons that involve equality $(\leq, \geq, =)$ become dubious and should be avoided, i.e., avoid using closed $[\cdot,\cdot]$ or half-closed $[\cdot,\cdot)$, $(\cdot,\cdot]$ intervals. On the other hand, if the timing constraints refer to the number of samples, then the bounds on `I` are stored using integer variables and comparisons involving equalities are meaningful.

Line 06 refers to the dimension n of the space $\mathbb{R}^n$ that the signal takes values. Version 0.1 of TaLiRo only supports 1D signals, i.e., $n = 1$. This line is added for compatibility with future versions.

Line 08 refers to the number of predicates which are in the domain of the observation map $\mathcal{O}$. The following lines (09-19) contain the definition of each set that is labeled by an atomic proposition. The declaration of each set begins with the statement :

[3] In future versions of TaLiRo, if no temporal operator in the formula has timing constraints, then the formula will be tested using a more memory efficient LTL version of the algorithm.

```
<predicate> number of constraints : <m>
```

In the position of `<predicate>`, we can place any predicate name that is a combination of alphanumeric characters, e.g., `pred1`, `p1`, `bb`, `aa123bb` etc. For computational reasons, the subsets of $\mathbb{R}^n$ are defined using intersections of halfspaces. This implies that the sets labeled by the atomic propositions are actually convex polyhedral sets. Note that this is not a fundamental restriction of the toolbox, since concave sets can be defined by taking the negation of an atomic proposition. The number `m` denotes how many halfspaces define a set. Each halfspace i of the set is represented by an inequality of the form $\sum_{j=1}^{n} a_{ij}\sigma^j \leq b_i$, where σ^j is the projection of the signal σ on its j dimension (or variable) and $a_{ij}, b_i \in \mathbb{R}$. Then, the set $\mathcal{O}(<\texttt{predicate}>)$ can be defined by the conjunction of the aforementioned inequalities : $\bigwedge_{i=1}^{m} \sum_{j=1}^{n} a_{ij}\sigma^j \leq b_i$. The latter can also be represented by a matrix inequality $A\sigma \leq B$, where $A = \{a_{ij}\}$ and $B = \{b_i\}$. The matrices A and B are given as inputs to TaLiRo in the form of a concatenated matrix $[A|B]$. For example, consider the atomic proposition `p2` defined in lines 13-15 which has two constraints. The first constraint indicates that $\sigma^1 \leq 0.5$, while the second that $-\sigma^1 \leq 0.5$. In other words, $\mathcal{O}(\texttt{p2}) = [-0.5, 0.5]$. In lines 10-11, the atomic proposition `p1` defines the set $\mathcal{O}(\texttt{p1}) = [1, +\infty)$.

Version 0.1 of TaLiRo does not check for emptiness of the sets. Future versions will have this functionality.

Line 20 indicates whether the timing constraints refer to the number of sampling points or not. In cases where the sampling step is constant, i.e., $\tau(i+1) - \tau(i) = \Delta\tau \in \mathbb{Q}$ for all $i > 0$, then it might be beneficial to write the timing constraints on the temporal operators with respect to the number of sampling points instead the actual time. For example, if $\Delta\tau = 0.1$, then the formula `<>_[0.1,0.5]p1` can be converted to `<>_[1,5]p1`. In the latter case, the equality checks become meaningful since we require that `p1` holds at some point between the next and the next five samples.

If the answer in line 20 is `no` or the temporal logic formula is in LTL, then we must still provide the timestamps in the file `inputdata.txt` even though the timestamps are ignored by the algorithm.

Finally, line 21 indicates the length of the input timed state sequence.

The file `inputdata.txt` contains the timestamps and the data of the discrete time signal. The first column contains the timestamps of the samples, while the following columns contain the data for each dimension of the signal. The following table presents the first 5 lines of such an input file generated for an 1D signal. In this example, the sampling step is constant with $\Delta\tau = 0.01$.

```
0.0000000e+000 0.0000000e+000
1.0000000e-002 2.3998800e-002
2.0000000e-002 4.7990401e-002
3.0000000e-002 7.1967605e-002
4.0000000e-002 9.5923223e-002
       ⋮
```

3.2 Examples

The examples presented below were run on PIII 1.2GHz with 1GB RAM under Windows XP. First, assume that we are given the discrete time signal σ_1 (see Fig. 4) and the corresponding timing function τ_1. The signal σ_1 has 110 sampling points. We would like to verify that whenever the value of the signal raises above the value 1.5, then it drops below 1.5 within 1 time unit. This can be formally stated with the MTL formula

```
[](p1 -> <>_(0.0,1.0)!p1)
```
(1)

where $\mathcal{O}(\texttt{p1}) = [1.5, +\infty)$. The output of TaLiRo for this case is

```
robustness : 0.097603
total running time : 0.030000 sec
```

In this example, since the sampling step is constant, i.e., 0.2, we can test the same specification over the number of samples and include the upper bound of the timing constraint – if this is desirable. Then, the formula becomes

```
[](p1 -> <>_(0,5]!p1)
```
(2)

The output of TaLiRo is

```
robustness : 0.317274
total running time : 0.020000 sec
```

If we reduce the timing constraint to 0.5, i.e., the MTL formula is

```
[](p1 -> <>_(0.0,0.5)!p1)
```
(3)

then the specification does not hold any more (`robustness : -0.158058`). What if we don't only want the signal value to drop below 1.5, but also to stay below 1.5 for at least 2 time units? Then, we can use the MTL formula

```
[](p1 -> <>_(0,5)[]_[0,10]!p1)
```
(4)

where the constraints are on the number of samples. The result is

```
robustness : 0.097603
total running time : 0.140000 sec
```

Now, if we increase the *always* bounds from 2 time units to 10, that is,

```
[](p1 -> <>_(0.0,1.0)[]_(0.0,10.0)!p1)
```
(5)

where the constraints are on the actual time, then the specification does not hold (`robustness : -0.250768`). Next, assume that we would like to test whether the signal oscillates between the sets `p2` and `p1`, where $\mathcal{O}(\texttt{p2}) = (-\infty, -1.5]$, in that order. The LTL formula

```
[](<>(p2/\<> p1))
```
(6)

signal's time domain	number of samples	computation time (sec)	robustness
$[0, 21.99]$	110	0.00	0.097603
$[0, 188.49]$	943	0.03	0.097603
$[0, 6283.2]$	31,416	1.05	0.092065
$[0, 219911.48]$	1,099,558	37.61	0.91793

Table 2. Computation time for formula `[](p1 -> <>_(0.0,1.0)!p1)`.

does not do the job. Even though the signal is periodic and event `p1` follows `p2`, the robustness of the formula is -1.683066. Here, by `p` event we mean that the value of the signal is within the set $\mathcal{O}$(`p`). This situation occurs because the signal is of finite duration and the *always* operator in formula 6 requires that that the subformula `<>(p2 /\ <> p1)` holds at the last sample of the signal (which is obviously not true since there are no more sampling points). If the period of the signal can be estimated, then we could use that as an upper bound on the *always* operator. For example, for signal σ_1 the period is 2π so we could instead test the formula

```
[]_[0.0,12.57)(<>(p2/\<> p1))        (7)
```

which is correct with robustness 0.238435. This example implies that we should sample or simulate the discrete time signal for some time that is longer than the time interval that we would like to test for periodicity. In addition, we can add constraints on the occurrence of the events. For example, for the input formula

```
[]_[0.0,12.57)(<>_[0.0,6.28)(p2/\<>_[0.0,3.14) p1))        (8)
```

we get

```
robustness : 0.238435
total running time : 0.781000 sec
```

Note that formulas (6), (7) and (8) ignore (i) the fact that initially the event `p1` happens before the event `p2` and (ii) that the signal σ_1 can take values inside and outside the set $\mathcal{O}$(`p2`) several times before event `p1` is observed.

Tables 2 and 3 indicate how the computation time of TaLiRo changes as the length of the discrete time signal increases. The sampling function is again τ_1. For these experimental data, the computer used was a Dell PowerEdge 1650 with two 1.4GHz Pentium-III CPUs and 2GB of RAM running SUSE 9.2 Linux. We can see that the computation time and the memory requirements do not only depend on the length of the discrete time signal, but also on the structure of the formula. A heuristic way to reduce the computation time and the memory requirements of TaLiRo is to add timing constraints on the temporal operators whenever this is possible. Notice that in Tables 2 and 3 the robustness is not constant with respect to the length of the timed state sequence. This happens because the sampling step is 0.2 and, thus, in each period the samples of the signal that belong to the set $\mathcal{O}$(`p1`) are different.

An immediate application of the robustness estimate is the following. Assume that σ_1 is not generated as indicated in Fig. 4, but that it is the result of sampling

signal's time domain	number of samples	computation time (sec)	robustness
$[0, 21.99]$	110	0.16	0.238435
$[0, 188.49]$	943	2.35	0.237401
$[0, 6283.2]$	31,416	84.74	0.237149
$[0, 219911.48]$	1,099,558	out of memory	-

Table 3. Computation time for formula (8) for the first row and for formula `[]_[0.0,T)(<>_[0.0,6.28)(p2/\<>_[0.0,3.14)p1))` for the rest of the rows, where `T` is the maximum signal time minus 3π.

(monitoring) a physical quantity. In such a case, the sensors, which monitor the quantity, have a known experimentally determined accuracy. As an example, assume that the accuracy of the sensor in our case is ± 0.1. Then, we immediately can infer that formulas (2), (7) and (8) are true over the monitored sampled signal since 0.1 (the sensor accuracy) is less then the robustness estimate of the formulas : 0.317274, 0.238435 and 0.238435 respectively. On the other hand, we cannot infer whether the signal σ_1 satisfies formulas (1) and (4) since 0.1 is greater then their robustness estimate of 0.097603 with respect to σ_1. If we would like to logically infer something about the underlying continuous time signal (not the sampled one), then one way to do so is to use the approach which was proposed in [11].

Now consider the following scenario. Signal s_1 is fed into a system which tries to track the input signal. The output of the system is signal s_2 in Fig. 6. For the shake of example, we set s_2 to be s_1 with a delay of 0.1 time units and a bounded noise of 0.1 magnitude. We monitor both signals with a constant sampling step of 0.2 time units. The result of the sampling process appears in Fig. 7. We would like to verify whether σ_2 is always within distance 0.25 of signal σ_1 and, moreover, if the difference of the two signals is greater than 0.25, then it should drop below 0.25 within 1 time unit. The above informal specification can be formally captured with the MTL formula $\Box(p_3 \vee (\neg p_3) \rightarrow \Diamond_{[0,1]} p_3)$ with $\mathcal{O}(p_3) = (-\infty, 0.25]$, which can be simplified to the formula $\Box(p_3 \vee \Diamond_{[0,1]} p_3)$. Now, if we test formula

```
[](p3\/<>_[0,5]p3)        (9)
```

where the timing constraints are on the number of samples, over the signal $\sigma_3(i) = |\sigma_1(i) - \sigma_2(i)|$, we get

```
robustness : 0.038030
total running time : 0.020000 sec
```

If the timing constraints are stricter, that is,

```
[](p3\/<>_[0,2]p3)        (10)
```

then the property does not hold any more (robustness -0.046525).

Acknowledgments This work has been partially supported by NSF EHS 0311123, NSF ITR 0324977 and ARO MURI DAAD 19-02-01-0383.

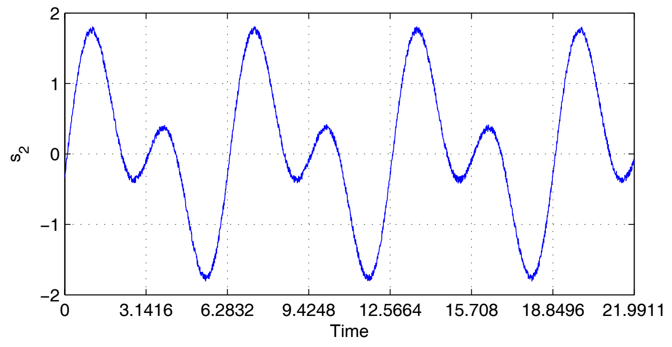

Fig. 6. The signal s_2.

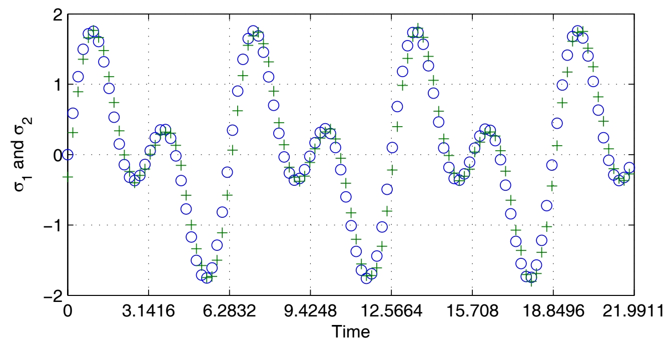

Fig. 7. The samples of signal σ_1 are denoted by circles (o), while the samples of signal σ_2 by crosses (+).

References

1. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge, Massachusetts (1999)
2. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theoretical Computer Science **138** (1995) 3–34
3. Kapinski, J., Krogh, B.H., Maler, O., Stursberg, O.: On systematic simulation of open continuous systems. In: Hybrid Systems: Computation and Control. Volume 2623 of LNCS., Springer (2003) 283–297
4. Esposito, J.M., Kim, J., Kumar, V.: Adaptive RRTs for validating hybrid robotic control systems. In: Proceedings of the International Workshop on the Algorithmic Foundations of Robotics. (2004)
5. Tan, L., Kim, J., Sokolsky, O., Lee, I.: Model-based testing and monitoring for hybrid embedded systems. In: Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration. (2004) 487–492
6. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Proceedings of FORMATS-FTRTFT. Volume 3253 of LNCS. (2004) 152–166

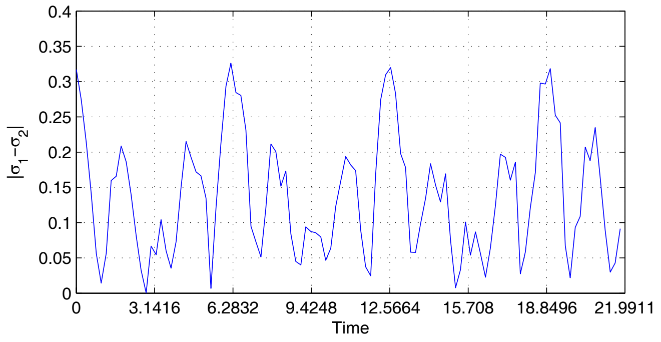

Fig. 8. The discrete time signal $\sigma_3(i) = |\sigma_1(i) - \sigma_2(i)|$.

7. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications. In: Proceedings of FATES/RV. Volume 4262 of LNCS., Springer (2006) 178–192
8. Emerson, E.A.: Temporal and modal logic. In van Leeuwen, J., ed.: Handbook of Theoretical Computer Science: Formal Models and Semantics. Volume B. North-Holland Pub. Co./MIT Press (1990) 995–1072
9. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems **2** (1990) 255–299
10. Fainekos, G.E., Girard, A., Pappas, G.J.: Temporal logic verification using simulation. In: Proceedings of FORMATS. Volume 4202 of LNCS., Springer (2006) 171–186
11. Fainekos, G.E., Pappas, G.J.: Robust sampling for MITL specifications. In: Proceedings of FORMATS. Volume 4763 of LNCS., Springer (2007) 147–162
12. Alur, R., Henzinger, T.A.: Real-Time Logics: Complexity and Expressiveness. In: Fifth Annual IEEE Symposium on Logic in Computer Science, Washington, D.C., IEEE Computer Society Press (1990) 390–401
13. Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., Schnoebelen, Ph.: Systems and Software Verification. Model-Checking Techniques and Tools. Springer (2001)
14. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th IEEE Symposium Foundations of Computer Science. (1977) 46–57
15. Fainekos, G.E.: Robustness of Temporal Logic Specifications. PhD thesis, Department of Computer and Information Science, University of Pennsylvania (2008)
16. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for finite state sequences in metric spaces. Technical Report MS-CIS-06-05, Dept. of CIS, Univ. of Pennsylvania (2006)