On-Line Monitoring for Temporal Logic Robustness

Adel Dokhanchi, Bardh Hoxha, and Georgios Fainekos

School of Computing, Informatics and Decision Systems Engineering
Arizona State University
{adokhanc,bhoxha,fainekos}@asu.edu

Abstract. In this paper, we provide a Dynamic Programming algorithm for on-line monitoring of the state robustness of Metric Temporal Logic specifications with past time operators. We compute the robustness of MTL with unbounded past and bounded future temporal operators ($\text{MTL}_{+pt}^{<+\infty}$) over sampled traces of Cyber-Physical Systems. We implemented our tool in Matlab as a Simulink block that can be used in any Simulink model. We experimentally demonstrate that the overhead of the $\text{MTL}_{+pt}^{<+\infty}$ robustness monitoring is acceptable for certain classes of practical specifications.

1 Introduction

Modern airplanes, automobiles and medical devices are prime examples of safety critical Cyber-Physical Systems (CPS). Nowadays, the majority of safety critical functions in such systems is controlled by embedded computers. Due to the critical nature of these components, it is of paramount importance to verify the functional correctness of the embedded software. However, as the number of computer controlled components increases so does the complexity of the verification of functional correctness. Moreover, the verification problem of most classes of CPS is even an undecidable problem [1].

As an alternative to verification and off-line testing, runtime monitoring has been proposed. The underlying idea is that given a set of formal requirements, these requirements are analyzed at runtime by an independent monitor and if a violation is detected, it is reported to a supervisor. The supervisor can then decide on remedial actions to fix the problem or reduce its impact to the system. The monitoring problem has been extensively studied [2–14] for the cases where the formal requirements are expressed in Linear Temporal Logic (LTL) [15] or in Metric Temporal Logic (MTL) [16].

In this paper, we revisit the MTL runtime monitoring problem when targeted to CPS. In particular, we claim that the classical Boolean semantics (or even three valued semantics) are not sufficiently informative for CPS behaviors. For instance, consider the specification "*After a takeoff command is received, then reach altitude of 600ft within 5 minutes*" for an autonomous Unmanned Aerial Vehicle (UAV) as introduced in [8]. Clearly, knowing that the specification failed or passed at runtime is important. However, more useful information from the perspective of the supervisor would be the knowledge of how far is the aircraft from satisfying the requirement. More specifically, -10ft from the requirement of 600ft at 1 min away from the 5 min threshold should potentially be less alarming than -100ft at exactly the same time. A supervisor that has a model of the dynamics of the aircraft can determine whether the UAV can climb 100ft within 1 min or not. We remark that the determination of the climb rate can only occur at runtime since this depends on the atmospheric parameters, the payload of the UAV, etc. Hence, the climb rate cannot be a precomputed parameter unless it is very conservatively set.

Our goal is to construct MTL monitors for estimating the robustness of satisfaction [17–19]. Temporal logic robustness gives a quantitative interpretation of satisfaction of an MTL formula. In detail, if an MTL formula valuates to positive robustness ε, then the specification is true and, moreover, the state sequences can tolerate perturbations up to ε and still satisfy the specification. Similarly, if the robustness is negative, then the specification is false and, moreover, the state sequences under ε perturbations still do not satisfy the specification. Thus, robust semantics can be used to give quantitative values to the satisfaction of MTL formulas when the target is CPS.

The challenge here is that automata based monitors [13, 14] cannot be synthesized for computing the robustness valuations. Therefore, formula rewriting methods [11] or dynamic programming [9] methods must be used. Here, we take the latter approach for combined unbounded past time and bounded future time MTL specifications. Since we are working with CPS, we assume that it is possible - if desired - to have a model predictive component in the system [20] which will provide a finite horizon prediction of the system behavior. That finite horizon prediction could be appended with the observed system behavior to provide a robustness estimate of a likely system behavior. Hence, it becomes possible to monitor specifications such as "*If at anytime in the past a take-off command is issued, then within 5 min the altitude of 600ft is reached*". Thus, such requirements can now be monitored using only the actual observed system behavior or the observed system behavior with the predicted system behavior.

Our contributions in this paper are as follows: We provide a dynamic programming algorithm for on-line monitoring of the robustness metric of MTL formulas with bounded future and unbounded past. In addition, we provide a Matlab/Simulink toolbox that can be used in any Simulink model for runtime monitoring of MTL robustness. The memory usage of our method is bounded and its runtime overhead is negligible for practical applications. Additional benefits in utilizing an on-line monitor are that it can be used in temporal logic testing algorithms [21, 22], where it may be desirable that the simulation stops as soon as the property is violated, as well as in feedback control for MTL specifications. Although temporal logic robustness has been considered in previous works [17–19], the solutions were provided for off-line testing. To the best of our knowledge, this is the first attempt to solve the on-line MTL robustness monitoring problem efficiently.

2 Problem Formulation

In the following, we represent the set of natural numbers including zero by $\mathbb{N}$ and the finite interval of $\mathbb{N}$ up to m by $\mathbb{N}_m = \{0, 1, \ldots, m\}$. In this work, we consider monitoring of Cyber-Physical Systems (CPS). We assume that we have access to some discrete time execution or simulation traces of the CPS. We view *(execution or simulation) traces* as timed state sequences $\mathcal{T} = \mathcal{T}_0\mathcal{T}_1\mathcal{T}_2 \ldots \mathcal{T}_m = (\tau_0, s_0)\ (\tau_1, s_1)\ (\tau_2, s_2) \ldots (\tau_m, s_m)$ where for each $k \in \mathbb{N}_m$, $\tau_k \in \mathbb{R}_{\geq 0}$ is a time stamp and $s_k \in S$ is a vector containing the

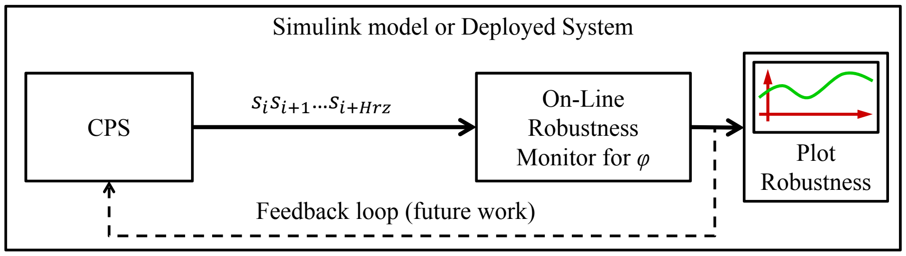
Simulink model or Deployed System
CPS
$s_i s_{i+1} \dots s_{i+Hrz}$
On-Line Robustness Monitor for φ
Plot Robustness
Feedback loop (future work)

Fig. 1. Overview of the solution of the $\text{MTL}_{+pt}^{<+\infty}$ on-line monitoring problem. The monitored robustness values could be used as feedback to the CPS or it could be plotted to be observed by a human supervisor if needed.

values of the state variables of the system at each sampling instance k. For example, for $m = 2$, the trace $\mathcal{T} = (0, (2, 0.34))(0.1, (3, 0.356))(0.2, (2, 0.36))$ captures the finite time execution of a CPS with two state variables in the vector s_k: one ranging over the natural numbers $\mathbb{N}$ and the other over the reals $\mathbb{R}$. That is, for $k = 1$, the state of the system at time $\tau_1 = 0.1$ was $s_1 = (3, 0.356) \in \mathbb{N} \times \mathbb{R}$. We further assume that $\mathcal{S} = (S, d)$ is a generalized quasi-metric space [23]. The existence of metrics is necessary so that distances can be defined for quantitative valuations of the atomic propositions [18, 21].

Throughout the paper, the variable i, which ranges over $\mathbb{N}$, is used to represent the current simulation step or the current index of the sampling process. We assume a fixed sampling period for the monitored system. Thus, there exists a fixed time period between consecutive time stamps. For this fixed time period $\varDelta t > 0$, for all $i \geq 0$, we have $\tau_{i+1} - \tau_i = \varDelta t$ (or equivalently $\tau_i = i\varDelta t$). As a result, we can simply compute each time stamp τ_i knowing the trace index (or simulation step) i by multiplication ($\tau_i = i\varDelta t$). Therefore, we use the trace index (simulation step i) as the reference of time.

The property of interest is stated in Metric Temporal Logic (MTL) with bounded future and unbounded past ($\text{MTL}_{+pt}^{<+\infty}$) for timed state sequences [11]. More specifically, at each time i, we would like to monitor safety requirements represented as $\text{MTL}_{+pt}^{<+\infty}$ formulas. These formulas capture safety properties of the system, such as bounded reactivity, which can be periodically analyzed for violation. In our formulation, we use the robust (quantitative) semantics [18] that quantify the distance between a given execution trace of a CPS and all the execution traces that violate the property. The robustness of a formula $[\![\varphi]\!]$ with respect to a trace $\mathcal{T}$ at time i is a value that measures how far is the trace from the satisfaction/falsification. This measure is an extension of boolean values representing satisfaction or falsification which is used in conventional monitoring. A positive robustness value means that the trace satisfies the property and a negative robustness means that the specification is not satisfied.

Our goal in this paper is to provide monitoring tools for temporal logic robustness. We assume that at each time i, the CPS outputs its current state s_i along with a finite prediction s_{i+1}, s_{i+2}, $\dots$, s_{i+Hrz} of horizon length $Hrz \in \mathbb{N}$ (see Fig. 1). The horizon length Hrz will be formally defined in Sec. 4; however, informally, it is the required number of samples after time i so that any future requirements in the MTL specification ϕ are resolved, i.e., the horizon depends on the structure of the formula ϕ, $Hrz = hrz(\phi)$.

When dealing with CPS, there exist numerous methods by which such a prediction horizon (forecasting) can be computed [24–26].

Next, we formally define the main problem presented in this paper.

Problem 1 (MTL$_{+pt}^{<+\infty}$ Robustness Monitoring) *Given an MTL$_{+pt}^{<+\infty}$ specification φ, a sampling instance i and an execution trace $\mathcal{T} = \mathcal{T}_0\mathcal{T}_1 \dots \mathcal{T}_m$ such that $m = i + hrz(\varphi)$, compute the current robustness estimate $[\![\varphi]\!](\mathcal{T}, i)$ at time τ_i.*

Intuitively, φ represents a system invariant that must hold at every point in the system execution. This can also be viewed as testing for the specification robustness $[\![\Box\varphi]\!](\mathcal{T}, 0)$, where $\Box$ is the operator for "*always in the future*" and φ is an arbitrary MTL$_{+pt}^{<+\infty}$ specification. However, instead of caring about the satisfaction of the formula at the beginning of the time, we care about the potential of violating φ for which we design an on-line monitor.

Overview of solution and summary of contributions: We provide an on-line monitoring approach for computing the robustness of an MTL$_{+pt}^{<+\infty}$ formula with respect to execution traces of a CPS. An overview of the solution for the MTL$_{+pt}^{<+\infty}$ on-line monitoring problem appears in Fig. 1. Our method monitors the behavior of a CPS as it executes. Our toolbox is also useful for applications where Simulink models are actually used for process monitoring (and not simulation). In addition, it can also be used for code generation for general MTL$_{+pt}^{<+\infty}$ monitors for deployment on actual systems. Our method computes the robustness of invariants $[\![\varphi]\!](\mathcal{T}, i)$ by storing previous specification robustness values – if needed – and by only utilizing a bounded number of pairs of the execution trace $\mathcal{T}_{Hst}, \dots, \mathcal{T}_{Hrz}$ where $Hst \in \mathbb{N}_i$ and it will be formally defined in Sec. 4. Our monitor uses bounded memory and, in the worst case, it has quadratic time complexity that depends on the magnitude of $Hrz - Hst$. In principle, our solution for robustness monitoring is inspired by the boolean temporal logic monitoring algorithm in [2].

3 Robustness of Metric Temporal Logic Specifications

In digital control and monitoring of CPS, it is inevitable that physical quantities are measured through a sampling process. As mentioned in the Problem Formulation section, when we mention time, we are actually referring to the corresponding sampling index i. With a slight abuse of notation and under the assumption of constant sampling rate, an execution trace $\mathcal{T}$ can also be represented by a function $s : \mathbb{N}_{i+Hrz} \to S$. The view of the sequence $s_0 s_1 \dots s_{i+Hrz}$ as a function s simplifies the presentation of the robust semantics for MTL.

Using a metric d [23], we can define a distance function that captures how far away a point $x \in X$ is from a set $S \subseteq X$. Intuitively, the distance function assigns positive values when x is in the set S and negative values when x is outside the set S. The metric d must be at least a generalized quasi-metric as described in [21] which also includes the case where d is a metric as it was introduced in [18].

Definition 1 (Signed Distance). *Let $x \in X$ be a point, $S \subseteq X$ be a set and d be a metric. Then, we define the Signed Distance from x to S to be*

$$\mathbf{Dist}_d(x, S) := \begin{cases} -\inf\{d(x,y) \mid y \in S\} & \text{if } x \notin S \\ \inf\{d(x,y) \mid y \in X \backslash S\} & \text{if } x \in S \end{cases}$$

where inf is the infimum.

Metric Temporal Logic (MTL) was introduced by Koymans [16] to reason about the quantitative timing properties of boolean signals. In this paper, we use the standard fragment of MTL with bounded future, but also we allow the use of past time operators.

Definition 2 ($\mathbf{MTL}^{<+\infty}_{+pt}$ Syntax). *Let AP be the set of atomic propositions and $\mathcal{I}$ be any non-empty interval of $\mathbb{N}$, and $\overline{\mathcal{I}}$ be any non-empty interval of $\mathbb{N} \cup \{+\infty\}$. The set $MTL^{<+\infty}_{+pt}$ formulas is inductively defined as $\varphi ::= \top \mid p \mid \neg\varphi \mid \psi \vee \varphi \mid \psi \mathcal{U}_{\mathcal{I}} \varphi \mid \psi \mathcal{S}_{\overline{\mathcal{I}}} \varphi$ where $p \in AP$ and $\top$ stands for* true.

Note that we use the number of samples to represent the time interval constraints of temporal operators. For example assume that $\Delta t = 0.1$, then the MTL formula $\Diamond_{[0,0.5]} a$ where the timing constraints are over time is instead represented by $\Diamond_{[0,5]} a$ in $\text{MTL}^{<+\infty}_{+pt}$.

The propositional operators conjunction ($\wedge$) and implication ($\rightarrow$) are defined the usual way. All other bounded future temporal operators can be syntactically defined using Until ($\mathcal{U}_{\mathcal{I}}$), where $\bigcirc$ (Next), $\Diamond$ (Eventually), and $\Box$ (Always) are defined as $\bigcirc\varphi \equiv \top\mathcal{U}_{[1,1]}\varphi$, $\Diamond_{\mathcal{I}}\varphi \equiv \top\mathcal{U}_{\mathcal{I}}\varphi$, and $\Box_{\mathcal{I}}\varphi \equiv \neg\Diamond_{\mathcal{I}}\neg\varphi$ respectively. The intuitive meaning of the $\psi\mathcal{U}_{[a,b]}\varphi$ operator at sampling time i is a follows: ψ has to hold at least until φ becomes true within the time interval of $[i+a, i+b]$ in the future. Similarly, all other bounded/unbounded past temporal operators can be defined using Since ($\mathcal{S}_{\overline{\mathcal{I}}}$), where $\odot$ (Previous), $\Diamond\!\!\!\cdot$ (Eventually in the past), and $\boxdot$ (Always in the past) are defined as $\odot\varphi \equiv \top\mathcal{S}_{[1,1]}\varphi$, $\Diamond\!\!\!\cdot_{\overline{\mathcal{I}}}\varphi \equiv \top\mathcal{S}_{\overline{\mathcal{I}}}\varphi$, and $\boxdot_{\overline{\mathcal{I}}}\varphi \equiv \neg\Diamond\!\!\!\cdot_{\overline{\mathcal{I}}}\neg\varphi$ respectively. The intuitive meaning of the $\psi\mathcal{S}_{[a,b]}\varphi$ operator at sampling time i is as follows: since φ becomes true within the interval $[i-b, i-a]$ in the past, ψ must hold till now (current time i).

$\text{MTL}^{<+\infty}_{+pt}$ can state requirements over the observable trajectories of a CPS. In order to capture these requirements, each predicate $p \in AP$ is mapped to a subset of the metric space X. We use an observation map $\mathcal{O}$ to interpret each predicate $p \in AP$. In other words, the observation map is defined as $\mathcal{O} : AP \rightarrow P(X)$ such that for each $p \in AP$ the corresponding set is $\mathcal{O}(p)$. Here, $P(S)$ denotes the powerset of a set S. We define the robust valuation of an $\text{MTL}^{<+\infty}_{+pt}$ formula φ over a trace s as follows [17].

Definition 3 ($\mathbf{MTL}^{<+\infty}_{+pt}$ Robustness Semantics). *Let s be a trace $s : \mathbb{N} \rightarrow X$, and $\mathcal{O}$ be an observation map $\mathcal{O} : AP \rightarrow P(X)$, then the robust semantics of any formula $\varphi \in$*

$MTL^{<+\infty}_{+pt}$ with respect to s is recursively defined as:

$$\begin{aligned}
[\![\top]\!](s,i) &:= +\infty\\
[\![p]\!](s,i) &:= \mathbf{Dist}_d(s(i), \mathcal{O}(p))\\
[\![\neg\varphi]\!](s,i) &:= -[\![\varphi]\!](s,i)\\
[\![\psi \vee \varphi]\!](s,i) &:= [\![\psi]\!](s,i) \sqcup [\![\varphi]\!](s,i)\\
[\![\psi \mathcal{U}_{[l,u]} \varphi]\!](s,i) &:= \bigsqcup_{j=i+l}^{i+u} \Big([\![\varphi]\!](s,j) \sqcap \bigsqcap_{k=i}^{j-1} [\![\psi]\!](s,k) \Big)\\
[\![\psi \mathcal{S}_{[l',u'\rangle} \varphi]\!](s,i) &:= \bigsqcup_{j=max\{0,i-u'\}}^{i-l'} \Big([\![\varphi]\!](s,j) \sqcap \bigsqcap_{k=j+1}^{i} [\![\psi]\!](s,k) \Big)
\end{aligned}$$

where $\sqcup$ stands for max, $\sqcap$ stands for min, $p \in AP$, $l,u,l' \in \mathbb{N}$ and $u' \in \mathbb{N} \cup \{\infty\}$. Furthermore, the symbol $\rangle$ in $\mathcal{S}_{[l',u'\rangle}$ will be) when $u' = +\infty$ and] when $u' \neq +\infty$.

We should point out that we use the extended definition of maximum ($\sqcup$) and minimum ($\sqcap$), with slight abuse of notation, we let $\max(\emptyset) = -\infty$ and $\min(\emptyset) = +\infty$. i.e., over empty sets we treat min and max as infimum and supremum, respectively. For exact definition of infimum and supremum see [27].

4 Robustness Monitoring of MTL$^{<+\infty}_{+pt}$

4.1 Finite horizon and history of MTL$^{<+\infty}_{+pt}$

For each MTL$^{<+\infty}_{+pt}$ formula ψ we define the finite horizon $hrz(\psi)$ as the number of samples we need to consider in the future. In MTL, the satisfaction of the formula depends on what will happen in the future. In bounded MTL, the finite horizon $hrz(\psi)$ is the number of steps (samples) which we need to consider in the future in order to evaluate the formula ψ at the current time i. In other words, $hrz(\psi)$ is the number of steps into the future for which the truth value of the sub-formula ψ depends on [2]. Similarly, we define the finite history $hst(\psi)$ of ψ as the number of samples we need to look into the past. That is, the number of steps in the past for which the truth value of the sub-formula ψ depends on. Intuitively, the $hst(\psi)$ is the size of the history we need to consider in order to keep track of what happened in the past to evaluate the formula ψ at present time. The finite horizon and the history can be defined recursively. We define $hrz(\psi)$ (similar to $h(\psi)$ in [2]) and we add the recursive definition of $hst(\psi)$ in the following:

$hrz(p) = 0$ $\qquad$ $hst(p) = 0$

$hrz(\neg\psi) = hrz(\psi)$ $\qquad$ $hst(\neg\psi) = hst(\psi)$

$hrz(\psi \;\mathbf{OP}\; \varphi) = max\{hrz(\psi), hrz(\varphi)\}$ $\qquad$ $hst(\psi \;\mathbf{OP}\; \varphi) = max\{hst(\psi), hst(\varphi)\}$

$hrz(\psi \mathcal{U}_{[l,u]} \varphi) = max\{hrz(\psi) + u - 1, hrz(\varphi) + u\}$ $\qquad$ $hst(\psi \mathcal{U}_{[l,u]} \varphi) = max\{hst(\psi), hst(\varphi)\}$

$hrz(\psi \mathcal{S}_{[l',u'\rangle} \varphi) = max\{hrz(\psi), hrz(\varphi)\}$

$$hst(\psi \mathcal{S}_{[l',u'\rangle} \varphi) = \begin{cases} max\{hst(\psi) + u' - 1, hst(\varphi) + u'\} & \text{if } u' \neq +\infty \\ max\{hst(\psi) + l' - 1, hst(\varphi) + l'\} & \text{if } u' = +\infty \end{cases}$$

Table 1. Pre Vector and Robustness Table

Pre[k]	$T_{k,j}$	column $j \Rightarrow$	-2	-1	0	1	2
	row $k \Downarrow$	Time(i)	$i-2$	$i-1$	i	$i+1$	$i+2$
▨	$\psi_1 = \varphi$	$\psi_2 \wedge \psi_3$	$[\![\varphi]\!](s, i-2)$	$[\![\varphi]\!](s, i-1)$	$[\![\varphi]\!](s, i)$	$[\![\varphi]\!](s, i+1)$	$[\![\varphi]\!](s, i+2)$
▨	ψ_2	$\Box_{[1,2]} q$	$[\![\psi_2]\!](s, i-2)$	$[\![\psi_2]\!](s, i-1)$	$[\![\psi_2]\!](s, i)$	$[\![\psi_2]\!](s, i+1)$	$[\![\psi_2]\!](s, i+2)$
$[\![\psi_3]\!](s, i-3)$	ψ_3	$\boxdot_{[0,+\infty)} p$	$[\![\psi_3]\!](s, i-2)$	$[\![\psi_3]\!](s, i-1)$	$[\![\psi_3]\!](s, i)$	$[\![\psi_3]\!](s, i+1)$	$[\![\psi_3]\!](s, i+2)$
▨	ψ_4	p	$[\![\psi_4]\!](s, i-2)$	$[\![\psi_4]\!](s, i-1)$	$[\![\psi_4]\!](s, i)$	$[\![\psi_4]\!](s, i+1)$	$[\![\psi_4]\!](s, i+2)$
▨	ψ_5	q	$[\![\psi_5]\!](s, i-2)$	$[\![\psi_5]\!](s, i-1)$	$[\![\psi_5]\!](s, i)$	$[\![\psi_5]\!](s, i+1)$	$[\![\psi_5]\!](s, i+2)$

where $p \in AP$. Here, **OP** is any binary operator in propositional logic, and ψ, φ are $\mathrm{MTL}_{+pt}^{<+\infty}$ formulas. For the unbounded $\mathcal{S}_{[0,+\infty)}$ operator, the computation of finite history is more involved and needs more explanation. Namely, we need to restate the dynamic programming algorithm for monitoring a sub-formula $\psi \mathcal{S}_{[0,+\infty)} \varphi$ based on the following works [9, 5]. According to the robustness semantics, the robustness of $\psi \mathcal{S}_{[0,+\infty)} \varphi$ at time i is as follows:

$$[\![\psi \mathcal{S}_{[0,+\infty)} \varphi]\!](s, i) = \bigsqcup_{j=0}^{i} \Big([\![\varphi]\!](s, j) \sqcap \bigsqcap_{k=j+1}^{i} [\![\psi]\!](s, k) \Big)$$

also robustness of $\psi \mathcal{S}_{[0,+\infty)} \varphi$ at time $i-1$ is

$$[\![\psi \mathcal{S}_{[0,+\infty)} \varphi]\!](s, i-1) = \bigsqcup_{j=0}^{i-1} \Big([\![\varphi]\!](s, j) \sqcap \bigsqcap_{k=j+1}^{i-1} [\![\psi]\!](s, k) \Big)$$

Thus, we can rewrite $[\![\psi \mathcal{S}_{[0,+\infty)} \varphi]\!](s, i)$ as

$$[\![\psi \mathcal{S}_{[0,+\infty)} \varphi]\!](s, i) = [\![\varphi]\!](s, i) \sqcup \Bigg([\![\psi]\!](s, i) \sqcap \Big(\bigsqcup_{j=0}^{i-1} \Big([\![\varphi]\!](s, j) \sqcap \bigsqcap_{k=j+1}^{i-1} [\![\psi]\!](s, k) \Big) \Big) \Bigg) =$$

$$= [\![\varphi]\!](s, i) \sqcup \Bigg([\![\psi]\!](s, i) \sqcap \Big([\![\psi \mathcal{S}_{[0,+\infty)} \varphi]\!](s, i-1) \Big) \Bigg)$$

Therefore, similar to [5] we recursively update the robustness of $\psi \mathcal{S}_{[0,+\infty)} \varphi$ at the current time i and save it in a variable called "*Pre*" to reuse it for the computation of the next time step (see [5] for more details). As a result, when we have an unbounded past time operator, we do not need the full history table. However, if the formula contains a nested future time operator, we need to extend the history to be long enough to contain the actual values. In other words, although for unbounded past time operators we do not need the whole history table, we should still extend the history to be able to store the actual simulation values (not the predicted values) in "*Pre*".

4.2 Robustness Computation Algorithm

For each $\mathrm{MTL}_{+pt}^{<+\infty}$ formula φ we construct a table called **Robustness Table** with width of $Hst + 1 + Hrz$, where $Hrz = hrz(\varphi)$ is the finite horizon of the specification formula φ, and, $Hst = Hrz + hst(\varphi)$, where $hst(\varphi)$ is the finite history of the specification φ. Hst is extended conservatively due to the fact that "*Pre*" value can only store the robustness

Algorithm 1 On-Line Monitor

Input: φ, $s'_i = s_i s_{i+1} \ldots s_{i+Hrz}$, **d**, O; **Global variables:** T, Pre; **Output**: $T_{1,0}$(*robustness value*).

```
    procedure MONITOR($\varphi, s'_i, \mathbf{d}, O$)
 1: for $k \leftarrow 1$ to $|\varphi|$ do
 2:     $Pre(k) \leftarrow T_{k,(-Hst+hst(\varphi_k))}$
 3: end for
 4: for $j \leftarrow 1 - Hst$ to $Hrz$ do
 5:     for $k \leftarrow 1$ to $|\varphi|$ do
 6:         if $\varphi_k = p \in AP$ then
 7:             $T_{k,j-1} \leftarrow T_{k,j}$
 8:         end if
 9:     end for
10: end for
11: for $k \leftarrow |\varphi|$ down to 1 do
12:     if $\varphi_k = \varphi_m \mathcal{S}_{[l',u']} \varphi_n$ then
13:         for $j \leftarrow -Hst$ to $Hrz$ do
14:             $T_{k,j} \leftarrow CR(\phi_k, j, s'_i, \mathbf{d}, O)$
15:         end for
16:     else
17:         for $j \leftarrow Hrz$ down to $-Hst$ do
18:             $T_{k,j} \leftarrow CR(\phi_k, j, s'_i, \mathbf{d}, O)$
19:         end for
20:     end if
21: end for
22: return $T_{1,0}$
    end procedure
```

values corresponding to the actual simulation. The height of the robustness table is the size of the formula φ ($|\varphi|$), where $|\varphi|$ is the number of sub-formulas of φ including itself. For example, assume we have a formula $\varphi = \boxdot_{[0,+\infty)} p \wedge \Box_{[1,2]} q$ and we intend to compute $[\![\varphi]\!](\mathcal{T}, i)$ at each time i. In formula φ, $Hst = 2$ and $Hrz = 2$. Since φ has unbounded past-time operators, it needs the Pre vector as well as the Robustness Table. The Pre vector appended to the Robustness Table is presented in Table 1. In particular, the Pre vector contains the value of past sub-formulas from the beginning of the time up to the current time.

In the following, we explain how the values of Table 2, the robustness table, are computed using Algorithms 1 and 2. In order to make our algorithms more readable, we used a vector to show the CPS output s_i, $s_{i+1}, \ldots, s_{i+Hrz}$ to the monitoring (see Fig. 1). We define a vector $s'_i = s_i s_{i+1} \ldots s_{i+Hrz}$ which appends current state s_i with predictions s_{i+1}, $s_{i+2}, \ldots, s_{i+Hrz}$. In Table 1, i is the current simulation step which corresponds to column 0. At each simulation step i, for each unbounded past time sub-formula ϕ, we first save the values of the column $-Hst + hst(\phi)$ in the Pre vector (Algorithm 1 lines 1-3) since the column $-Hst + hst(\phi)$ contains the robustness value of ϕ from the beginning of the simulation. We need the Pre vector to compute the robustness of ϕ at the next sampling time using the dynamic programming method. In the above example, for $\boxdot_{[0,+\infty)} p$ the value at column -2 is saved in Pre to be used during robustness computa-

Table 2. Robustness Computation of each table entries (Gray cells are unused)

$T_{k,j}$	$i-2$	$i-1$	i	$i+1$	$i+2$
$k \Downarrow, j \Rightarrow$	$j=-2$	$j=-1$	$j=0$	$j=1$	$j=2$
Pre[1]	$T_{2,-2} \sqcap T_{3,-2}$	$T_{2,-1} \sqcap T_{3,-1}$	$T_{2,0} \sqcap T_{3,0}$	$T_{2,1} \sqcap T_{3,1}$	$T_{2,2} \sqcap T_{3,2}$
Pre[2]	$T_{5,-1} \sqcap T_{5,0}$	$T_{5,0} \sqcap T_{5,1}$	$T_{5,1} \sqcap T_{5,2}$	$T_{5,2}$	$+\infty$
Pre[3]	Pre[3]$\sqcap T_{4,-2}$	$T_{3,-2} \sqcap T_{4,-1}$	$T_{3,-1} \sqcap T_{4,0}$	$T_{3,0} \sqcap T_{4,1}$	$T_{3,1} \sqcap T_{4,2}$
Pre[4]	$\mathbf{Dist}_d(s_{i-2}, O(p))$	$\mathbf{Dist}_d(s_{i-1}, O(p))$	$\mathbf{Dist}_d(s_i, O(p))$	$\mathbf{Dist}_d(s_{i+1}, O(p))$	$\mathbf{Dist}_d(s_{i+2}, O(p))$
Pre[5]	$\mathbf{Dist}_d(s_{i-2}, O(q))$	$\mathbf{Dist}_d(s_{i-1}, O(q))$	$\mathbf{Dist}_d(s_i, O(q))$	$\mathbf{Dist}_d(s_{i+1}, O(q))$	$\mathbf{Dist}_d(s_{i+2}, O(q))$

Algorithm 2 Robustness Computation (CR)

Input: φ_k, j, $s'_i = s_i s_{i+1} \ldots s_{i+Hrz}$, **d**, O; **Global variables:** T, Pre; **Output**: $T_{k,j}$.

```
    procedure CR($\varphi_k, j, s'_i, \mathbf{d}, O$)
 1: if $\varphi_k = \top$ then $T_{k,j} \leftarrow +\infty$
 2: else if $\varphi_k = p \in AP$ then
 3:     if $j >= 0$ then
 4:         $T_{k,j} \leftarrow Dist_d(s_{i+j}, O(p))$
 5:     end if
 6: else if $\varphi_k = \neg\varphi_m$ then
 7:     $T_{k,j} \leftarrow -T_{m,j}$
 8: else if $\varphi_k = \varphi_m \vee \varphi_n$ then
 9:     $T_{k,j} \leftarrow T_{m,j} \sqcup T_{n,j}$
10: else if $\varphi_m \mathcal{U}_{[l,u]} \varphi_n$ then
11:     if $j + l \leq Hrz$ then
12:         $tmp_{min} \leftarrow \bigsqcap_{j \leq j' < j+l} T_{m,j'}$
13:         $T_{k,j} = -\infty$
14:         for $j' \leftarrow j + l$ to $min\{Hrz, u + j\}$ do
15:             $T_{k,j} \leftarrow T_{k,j} \sqcup (tmp_{min} \sqcap T_{n,j'})$
16:             $tmp_{min} = tmp_{min} \sqcap T_{m,j'}$
17:         end for
18:     else
19:         $T_{k,j} = -\infty$
20:     end if
21: else if $\varphi_m \mathcal{S}_{[l',u']} \varphi_n$ then
22:     if $j - l' \geq -Hst$ then
23:         $tmp_{min} \leftarrow \bigsqcap_{j-l' < j' \leq j} T_{m,j'}$
24:         if $u' \neq +\infty$ then
25:             $T_{k,j} = -\infty$
26:             for $j' \leftarrow j-l'$ down to $max\{-Hst, j-u'\}$ do
27:                 $T_{k,j} \leftarrow T_{k,j} \sqcup (tmp_{min} \sqcap T_{n,j'})$
28:                 $tmp_{min} = tmp_{min} \sqcap T_{m,j'}$
29:             end for
30:         else
31:             if $j - hst(\varphi_k) = -Hst$ then
32:                 $tmp_{\mathcal{S}} \leftarrow Pre[k] \sqcap T_{m,j}$
33:             else
34:                 $tmp_{\mathcal{S}} \leftarrow T_{k,j-1} \sqcap T_{m,j}$
35:             end if
36:             $T_{k,j} \leftarrow (T_{n,j-l'} \sqcap tmp_{min}) \sqcup tmp_{\mathcal{S}}$
37:         end if
38:     else
39:         $T_{k,j} = -\infty$
40:     end if
41: end if
42: return $T_{k,j}$
    end procedure
```

tion. Then, we shift all the robustness table entries of the predicates by one position to the left (Algorithm 1, lines 4-10). Then the loop (Algorithm 1, lines 11-21) recursively calls Algorithm 2 to fill the robustness table for each sub-formula from bottom to top.

Each call of Algorithm 2 (CR) computes each table entry $T_{k,j}$ (see tables 1,2) where column j is the horizon/history index and row k is the sub-formula index. For past sub-formulas the table entries are computed from left to right (Algorithm 1, lines 12-15), and for future sub-formulas the table entries are computed from right to left (Algorithm 1, lines 16-19). New values for predicates (according to execution traces) will be placed in column 0 and the predicted values of the predicates will be saved in columns 1 to Hrz (Algorithm 2, lines 2-5). Table 2 shows the updates of predicate values in rows 4, and 5 which correspond to Algorithm 2, line 4.

In the following, we explain how the CR Algorithm 2 computes the MTL robustness values for three different cases of MTL:

Case 1 (Lines 10-20): The robustness of bounded future temporal sub-formulas with interval $[l, u]$ at each column j is computed given the values of its operands for columns j up-to $min\{j + u, Hrz\}$ (Line 14). For example, this case is used in table 2 to compute the robustness of sub-formula $\psi_2 = \Box_{[1,2]} q$ from right to left. Case 1 in CR Algorithm is similar to the DP-TALIRO algorithm [28].

Case 2 (Lines 24-29): The robustness of bounded past temporal sub-formulas with interval $[l', u']$ at each column j is computed given the values of its operands for columns j down-to $max\{j - u', -Hst\}$ (Line 26).

Case 3 (Lines 30-37): The robustness of unbounded past temporal sub-formulas with interval $[l', +\infty)$ for column j is computed using the stored value in column $j - 1$ in dynamic programming fashion (Line 34) and using the *Pre* vector (Line 32). For example, Case 3 is used to compute the robustness of $\psi_3 = \boxdot_{[0,+\infty]} p$ using $Pre[3]$ from left to right in table 2.

Finally, we update table entries for the top row which corresponds to $\psi_1 = \varphi$. Since its corresponding operator $\wedge$ is propositional (Algorithm 2 Lines 6-9), we can update its value from any direction. The high level explanation of Algorithm 1 is described as follows:

1. Store values of column $-Hst + hst(\phi_k)$ for each unbounded past sub-formula ϕ_k in $Pre[k]$ and shift the table entries of predicates to the left (Lines 1-10).
2. For each row i from $|\varphi|$ to 1 compute the robustness values according to:
   (a) If φ_i is a future temporal operator, for each column j from Hrz down to $-Hst$, update table entry $T_{i,j}$ using Algorithm 2.
   (b) If φ_i is a past temporal operator, for each column j from $-Hst$ up to Hrz update table entry $T_{i,j}$ using Algorithm 2.
3. Return the robustness ($T_{1,0}$).

5 Experimental Analysis and Case Studies

5.1 Runtime Overhead

First, we measure the overhead of the proposed monitoring framework on a slightly modified version of the Automatic Transmission (AT) model provided by Mathworks as a Simulink demo[1]. The experiments were conducted on a Windows 7, Intel Core2 Quad (2.99 GHz) with 8 GB RAM.

The physical model of the AT system has two continuous (real-valued) state variables which are also its monitored outputs: the speed of the engine ω and the speed of the vehicle v. The model includes an automatic transmission controller that exhibits both continuous and discrete behavior. It is a typical CPS model and specifications over both boolean and continuous variables can be formalized. However, since the valuation of the robustness of predicates over continuous state variables is computationally more expensive than a boolean valuation, we consider only specifications over continuous state variables for the impact analysis.

We introduce our $\text{MTL}_{+pt}^{<+\infty}$ monitoring block in the AT model and test the performance over a set of specifications. In order to test the runtime overhead of our work, we artificially generate 30 different $\text{MTL}_{+pt}^{<+\infty}$ formulas based on typical critical safety formulas to show that the runtime overhead depends on both of the size of the formula and the horizon/history. We test our method for 100 runs of monitoring algorithm for each specification (formula), and for each run we use 100 simulation steps. Then, we compute the mean and variance of the overhead for each simulation step which is the

[1] Available at: `http://www.mathworks.com/help/simulink/examples/modeling-an-automatic-transmission-controller.html`

Table 3. The overhead on each simulation step on the Automatic Transmission model with specifications of increasing length. Table entries are in milliseconds.

#	H=1,000				H=2,000				H=10,000			
	E		U		E		U		E		U	
	Mean	Var.	Mean	Var.	Mean	Var.	Mean	Var.	Mean	Var.	Mean	Var.
$\phi_1(H)$	2.39	0.00	4.83	0.00	8.03	0.00	15.8	0.001	188.8	0.001	358.5	0.036
$\phi_3(H)$	4.24	0.00	7.5	0.001	12.7	0.00	25.09	0.005	314.4	0.01	599	0.665
$\phi_5(H)$	4.66	0.00	8.36	0.001	14.01	0.00	27.8	0.005	309.2	0.077	650	0.014
$\phi_7(H)$	4.95	0.00	8.94	0.00	14.83	0.00	29.33	0.006	311	0.013	674.2	0.033
$\phi_9(H)$	5.23	0.00	9.46	0.001	15.4	0.001	30.56	0.007	317.5	0.011	683.5	0.698

execution time of Algorithm 1 in table 3. In this table, the overhead is measured on specifications that contain either nested Until operators (U columns) or nested Eventually operators (E columns).
We generate 30 formulas according to the following templates:

- **E formulas:** $\phi_n(H) = p_j \longrightarrow \psi_n(H/n)$
  where $H \in \mathbb{N}$ is the finite horizon of the formula. In table 3, we used 1,000, 2,000 and 10,000 for the size of the horizon. Here, p_j is an arbitrary predicate and $\psi_n(H/n)$ is defined recursively as follows:
  $\psi_1(h) = \Diamond_{[0,h]} p_k$ and $\psi_n(h) = \Diamond_{[0,h]}(p_l \wedge \psi_{n-1}(h))$, for $1 < n \leq 10$
  where $h = H/n$, i.e., the finite horizon H divided by the number of nested subformulas n and p_k, p_l are arbitrary predicates.
- **U formulas:** $\phi_n(H) = p_j \longrightarrow \psi_n(H/n)$
  where $H \in \mathbb{N}$ is the finite horizon of the formula. In table 3, we used 1,000, 2,000 and 10,000 for the size of the horizon of H. Here, p_j is an arbitrary predicate and $\psi_n(H/n)$ is defined recursively as follows:
  $\psi_1(h) = p_k \,\mathcal{U}_{[0,h]} p_l$ and $\psi_n(h) = p_m \,\mathcal{U}_{[0,h]}(p_n \wedge \psi_{n-1}(Y))$, for $1 < n \leq 10$
  where $h = H/n$ and p_k, p_l, p_m, p_m are arbitrary predicates.

As illustrated in table 3, the computational complexity of the monitoring algorithm is closely related to the horizon and history size. Since the algorithm's complexity is of order $O(n^2)$ where n is the horizon/history, the added overhead (in worst case execution) is quadratic in terms of the size of the horizon for some formulas in table 3 (like $\phi_1(H)$). Moreover, in most cases, the impact of the number of nested temporal operators is not significant compared to the size of horizon/history windows. From table 3, we notice that when the horizon and history size is less than 2,000, the overhead for each simulation step is negligible with our prototype implementation. Furthermore, for most practical reactivity requirements, it is quite unlikely that even a window size of 2,000 sampling points is necessary. Therefore, the method could be utilized in real world monitoring applications.

5.2 Case Study

In the following, we utilize the monitoring method on an industrial size high-fidelity engine model. The model is part of the SimuQuest Enginuity [29] Matlab/Simulink tool package. The Enginuity tool package includes a library of modules for engine component blocks. It also includes pre-assembled models for standard engine configurations. In this work, we use the Port Fuel Injected (PFI) spark ignition, 4 cylinder inline engine configuration. It models the effects of combustion from first physics principles on a cylinder-by-cylinder basis, while also including regression models for particularly complex physical phenomena. The model includes a tire-model, brake system model, and a drive train model (including final drive, torque converter and transmission). The input to the system is the throttle schedule. The output is the normalized air-to-fuel(A/F) ratio. Simulink reports that this is a 56 state model. Note that this number represents only the visible states. It is possible that more states are present in the blackbox s-functions which are not accessible. This is a high dimensional non-linear system for which reachability analysis is very difficult. It also includes lookup tables, non-linear components, and inputs that affect the switching guards.

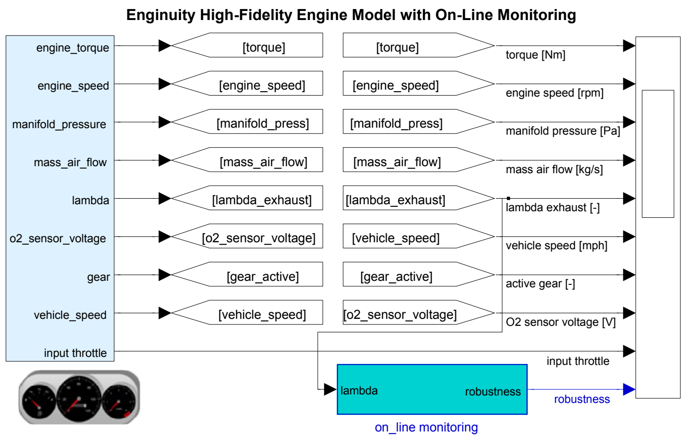

Fig. 2. SimuQuest [29] Enginuity Matlab Simulink engine model with the on-line monitoring block.

A specification of practical interest for an engine is the settling time for the A/F ratio, which is the quotient between the air mass and fuel mass flow. Ideally, the normalized A/F ratio λ should always be 1, indicating that the ratio of the air and fuel flow is the same as the stoichiometric ratio. Under engine operating conditions, this output fluctuates $\pm\%10$. We add the on-line monitoring block to the Simulink model as presented in Fig. 2.

Our goal is to monitor the engine while allowing temporary fluctuations to λ. We formally define the specification as follows:

$$\phi_{pt} = (\lambda \text{ out of bounds}) \rightarrow \Diamond_{[0,1]} \boxdot_{[0,1]} \neg(\lambda \text{ out of bounds})$$

Here, the formal specification states that if the A/F ratio exceeds the allowed bounds, then the ratio should have been settled for at least one second within the last two seconds.

Notice that an alternative presentation of the formula would be to use the future eventually and always operators, i.e. the formula would be defined as follows:

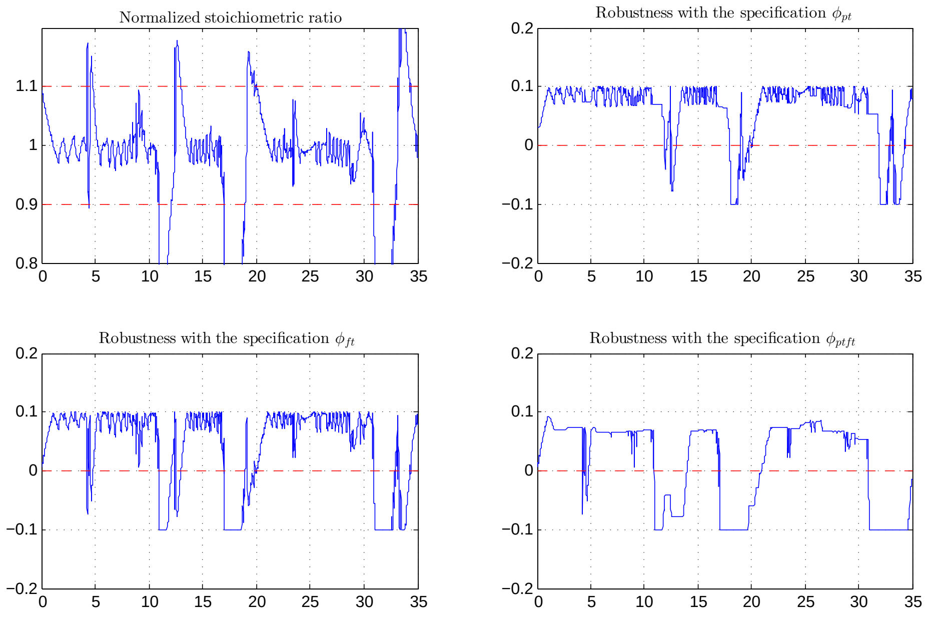

Fig. 3. Runtime monitoring of specifications ϕ_{pt}, ϕ_{ft} and ϕ_{ptft} on the high-fidelity engine model. The figure presents a normalized stoichiometric ratio, and the corresponding robustness values for specifications ϕ_{pt}, ϕ_{ft} and ϕ_{ptft}. Note that no predictor is utilized when computing the robustness values.

$$\phi_{ft} = (\lambda \text{ out of bounds}) \rightarrow \Diamond_{[0,1]}\Box_{[0,1]}\neg(\lambda \text{ out of bounds})$$

In this case, the specification states that always, if the A/F ratio output exceeds the allowed bounds, then within one second it should settle inside the bounds and stay there for a second.

Clearly, both ϕ_{pt} and ϕ_{ft} are equivalent in terms of the set of traces that satisfy/falsify each specification[2]. However, in real-time robustness monitoring, there is an important distinction between the two. When the specification requires future information, either a predictor is put in place or the semantics will handle only the current information. In this case, without a predictor, the future time formula reduces to the propositional formula $\phi_{ft} = (\lambda \text{ out of bounds}) \rightarrow \neg(\lambda \text{ out of bounds}) \equiv (\lambda \text{ out of bounds})$. Therefore, past time operators should be used. Recall that when monitoring robustness, our goal is to provide early warning on when the specification may fail by approaching dangerously an undesired threshold. In other words, the past formula allows us to reason about the robustness of the actual system observations, while the future formula in collaboration with a forecast model would allow us to estimate the likely robustness. This is in contrast to many boolean monitoring algorithms which issue an "*undecided until further notice*" verdict that does not provide any actionable information.

A third alternative monitoring specification is the following formula:

[2] Formally, this is the case if we ignore the first 2 seconds of the execution trace as well as the last 2 seconds – if the execution trace is finite.

$$\phi_{ptft} = \boxdot_{[0,2]}((\lambda \text{ out of bounds}) \rightarrow \Diamond_{[0,1]}\Box_{[0,1]}\neg(\lambda \text{ out of bounds}))$$

This specification states that at some point in the last two seconds, when λ is out of bounds then within the next second, λ will not be out of bounds and stay there for one second. This alternative seems to be the balance between the ϕ_{pt} and ϕ_{ft} formulas. Where ϕ_{pt} purely relies on past information, and ϕ_{ft} relies on information from a predictor, ϕ_{ptft} has the advantage that it utilizes both the information from the past but also it could include information from the predictor.

An example of real-time monitoring on the high-fidelity engine model is presented in Fig. 3. The figure illustrates the significance of using past time operators when defining specifications. Due to the lack of predictor information, the ϕ_{ft} monitor falsely returns falsification at about 4 seconds whereas the ϕ_{pt} monitor does not.

In the following, we analyze the overhead of the monitoring algorithm for this case study. Since the runtime is influenced by numerous sources of nondeterminism, we apply the central limit theorem to form confidence intervals for the mean simulation runtime when running the simulations with and without the monitor. To generate the results in table 4, we collected 30 samples with 100 simulation runtimes in each sample. We note that the difference between the estimated mean simulation runtime when adding the monitor is 0.97%. The experimental results were generated on an Intel Xeon X5647 (2.993GHz, 8 CPUs) machine with 12 GB RAM, Windows 7, and Matlab 2012a.

Table 4. Simulation runtime statistics for the high-fidelity engine model running for 35 seconds with simulation stepsize of 0.01s. The results include the confidence intervals for the mean simulation runtime.

Simulation runtime(sec.)	Est. Mean	Est. Std. Dev	95%		99%	
			LB	UB	LB	UB
Without monitor	10.811	0.090	10.778	10.844	10.766	10.857
With monitor	10.987	0.086	10.955	11.019	10.944	11.030

6 Conclusions and Future Work

We have presented an algorithm for monitoring the robustness of combined past and future MTL specifications. Our framework can incorporate predicted or estimated data as provided by a model predictive component. We have created a Simulink toolbox for MTL robustness monitoring which is distributed with the S-Taliro tools [30]. Our experiments indicate that the toolbox adds minimal overhead to the simulation time of Simulink models and it can be used for both runtime analysis of the models and for off-line testing. Our future work will concentrate on several aspects. First, the current version of the tool allows reasoning over timed state sequences generated under a constant sampling rate. We would like to relax this constraint so that we allow arbitrary sampling functions. Second, we would like to investigate the possibility of porting our monitor on FPGA platforms similar to [2, 8]. Finally, we envision that utilizing information about the system through the form of a model will permit us to move to an event based monitoring framework while still sufficiently approximating the robustness estimate.

Acknowledgments This work was partially supported by NSF awards CNS 1116136 and CNS 1319560. The authors would also like to thank the anonymous reviewers for the very detailed comments.

References

1. 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
2. Finkbeiner, B., Kuhtz, L.: Monitor circuits for ltl with bounded and unbounded future. In: Runtime Verification. Volume 5779 of LNCS., Springer (2009) 60–75
3. Havelund, K., Rosu, G.: Monitoring programs using rewriting. In: Proceedings of the 16th IEEE international conference on Automated software engineering. (2001)
4. Havelund, K., Rosu, G.: Synthesizing monitors for safety properties. In: Tools and Algorithms for the Construction and Analysis of Systems. Number 2280 in LNCS, Springer (2002) 342–356
5. Havelund, K., Rosu, G.: Efficient monitoring of safety properties. STTT **6** (2004) 158–173
6. Kristoffersen, K.J., Pedersen, C., Andersen, H.R.: Runtime verification of timed LTL using disjunctive normalized equation systems. In: Proceedings of the 3rd Workshop on Run-time Verification. Volume 89 of ENTCS. (2003) 1–16
7. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Proceedings of FORMATS-FTRTFT. Volume 3253 of LNCS. (2004) 152–166
8. Reinbacher, T., Rozier, K.Y., Schumann, J.: Temporal-logic based runtime observer pairs for system health management of real-time systems. In: Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Volume 8413 of LNCS., Springer (2014) 357–372
9. Rosu, G., Havelund, K.: Synthesizing dynamic programming algorithms from linear temporal logic formulae. Technical report, Research Institute for Advanced Computer Science (RIACS) (2001)
10. 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
11. Thati, P., Rosu, G.: Monitoring algorithms for metric temporal logic specifications. In: Runtime Verification. Volume 113 of ENTCS., Elsevier (2005) 145–162
12. Basin, D.A., Klaedtke, F., Zalinescu, E.: Algorithms for monitoring real-time properties. In: Runtime Verification. Volume 7186 of LNCS., Springer (2011) 260–275
13. Geilen, M.: On the construction of monitors for temporal logic properties. In: Proceedings of the 1st Workshop on Runtime Verification. Volume 55 of ENTCS. (2001) 181–199
14. Maler, O., Nickovic, D., Pnueli, A.: From MITL to Timed Automata. In: Proceedings of FORMATS. Volume 4202 of LNCS., Springer (2006) 274–289
15. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th IEEE Symposium Foundations of Computer Science. (1977) 46–57
16. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems **2** (1990) 255–299
17. Fainekos, G., Pappas, G.J.: Robustness of temporal logic specifications. In: Formal Approaches to Testing and Runtime Verification. Volume 4262 of LNCS., Springer (2006) 178–192
18. Fainekos, G., Pappas, G.J.: Robustness of temporal logic specifications for continuous-time signals. Theor. Comput. Sci. **410** (2009) 4262–4291

19. Donze, A., Ferrre, T., Maler, O.: Efficient robust monitoring for stl. In: Proceedings of the 25th International Conference on Computer Aided Verification. CAV, Berlin, Heidelberg, Springer-Verlag (2013) 264–279
20. Garcia, C.E., Prett, D.M., Morari, M.: Model predictive control: Theory and practice - a survey. Automatica **25** (1989) 335348
21. Abbas, H., Fainekos, G.E., Sankaranarayanan, S., Ivancic, F., Gupta, A.: Probabilistic temporal logic falsification of cyber-physical systems. ACM Trans. Embedded Comput. Syst. **12** (2013) 95
22. Jin, X., Donze, A., Deshmukh, J., Seshia, S.: Mining requirements from closed-loop control models. In: Hybrid Systems: Computation and Control, ACM Press (2013)
23. Seda, A.K., Hitzler, P.: Generalized distance functions in the theory of computation. The Computer Journal **53** (2008) bxm108443–464
24. Eklund, J.M., Sprinkle, J., Sastry, S.: Implementing and testing a nonlinear model predictive tracking controller for aerial pursuit/evasion games on a fixed wing aircraft. In: American Control Conference. (2005)
25. Bakirtzis, A., Petridis, V., Kiartzis, S., Alexiadis, M., Maissis, A.: A neural network short term load forecasting model for the greek power system. IEEE Transactions on Power Systems **11** (1996) 858–863
26. Monteiro, C., Bessa, R., Miranda, V., Botterud, A., Wang, J., Conzelmann, G.: Wind power forecasting: State-of-the-art 2009. Technical Report ANL/DIS-10-1, Argonne National Laboratory (2009)
27. Davey, B.A., Priestley, H.A.: Introduction to Lattices and Order (2. ed.). Cambridge University Press (2002)
28. Fainekos, G., Sankaranarayanan, S., Ueda, K., Yazarel, H.: Verification of automotive control applications using s-taliro. In: Proceedings of the American Control Conference. (2012)
29. Simuquest: Enginuity. `http://www.simuquest.com/products/enginuity` (2013) Accessed: 2013-10-14.
30. Annapureddy, Y.S.R., Liu, C., Fainekos, G.E., Sankaranarayanan, S.: S-taliro: A tool for temporal logic falsification for hybrid systems. In: Tools and algorithms for the construction and analysis of systems. Volume 6605 of LNCS., Springer (2011) 254–257