Measuring with Timed Patterns

Thomas Ferrère[1], Oded Maler[1], Dejan Ničković[2], and Dogan Ulus[1]

[1] VERIMAG, CNRS and the University of Grenoble-Alpes, France
[2] AIT Austrian Institute of Technology GmbH, Vienna, Austria

Abstract. We propose a declarative measurement specification language for quantitative performance evaluation of hybrid (discrete-continuous) systems based on simulation traces. We use timed regular expressions with events to specify patterns that define segments of simulation traces over which measurements are to be taken. In addition, we associate measure specifications over these patterns to describe a particular type of performance evaluation (maximization, average, etc.) to be done over the matched signal segments. The resulting language enables expressive and versatile specification of measurement objectives. We develop an algorithm for our measurement framework, implement it in a prototype tool, and apply it in a case study of an automotive communication protocol. Our experiments demonstrate that the proposed technique is usable with very low overhead to a typical (computationally intensive) simulation.

1 Introduction

Verification consists in checking whether system behaviors, sequences of states and events, satisfy some specifications. These specifications are expressed in a formalism, for example temporal logic, with well-defined semantics such that the satisfaction or violation of a property φ by a behavior w can be computed based on φ and w. To perform exhaustive formal verification, property φ is typically converted into an automaton $\mathcal{A}_{\neg\varphi}$ that accepts only violating sequences which is later composed with the system model and checked for emptiness. Such specifications are also used in a more lightweight and scalable form of verification (known as *runtime verification* in software and *assertion checking* in hardware) where *individual* behaviors are checked for property satisfaction. In this context, the formal specification language can be used to automatically derive *property monitors* rather than inspect execution traces manually or program monitors by hand. The specification formalism allows us to focus on the observable properties of the system we are interested in and write them in a *declarative* way, separated from their implementation. It is this concept that we export from the qualitative to the quantitative world.

Properties offer a purely *qualitative* way to evaluate systems and their behaviors: correct or incorrect. There are many contexts, however, where we want also to associate *quantitative* measures with systems and their executions. Consider for example a real-time system with both safety-critical and non-critical aspects, evaluated according to the temporal distance between pairs of *request*

and *service* events. Its safety-critical part will be evaluated according to whether some distance goes beyond a hard deadline. In contrast, its non-critical part is typically evaluated based on quality-of-service performance measures which are numerical in nature, such as the average response time or throughput.

Quantitative measures are used heavily in the design of cyber-physical systems involving heterogeneous components of computational and physical natures. Such systems exhibit continuous and hybrid behaviors and are often designed using modeling languages such as Simulink, Modelica or hardware description languages. These models are analyzed using a combination of numerical and discrete-event simulation, producing traces from which performance measures are extracted to evaluate design quality. Measures are computed by applying various operations such as summation/integration, arithmetical operations, max-min, etc. to certain segments of the simulation trace. The boundaries of these segments are defined according to the occurrence of certain *events* and *patterns* in the trace. When the measures are simple they are realized by inserting additional observer blocks to the system model but when they are more complex, they are extracted using manually-written (and error prone) procedural scripts that perform computations over the traces.

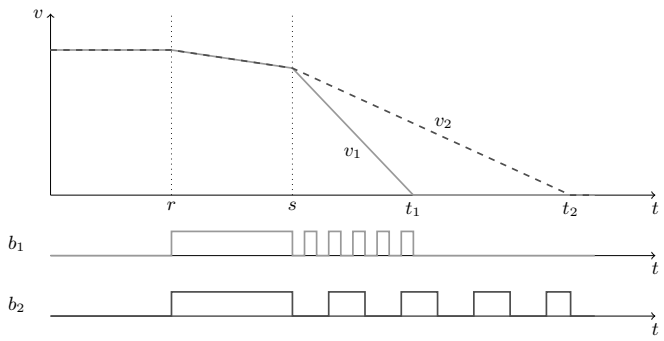


Fig. 1. Stopping distance measurement for anti-lock brake systems.

We illustrate how measurements can be used to compare two correct implementations of an anti-lock brake system (ABS), which prevents wheels from locking during heavy braking or on slippery roads. Figure 1 depicts braking control signals b_1 and b_2 and velocity signals v_1 and v_2 for two controller models C_1 and C_2. The driver starts to brake fully at $t = r$ and then the ABS takes control at $t = s$ and applies rapid pulsation to prevent locking. Both controllers C_1 and C_2 satisfy the anti-lock property but we also want to compare the distance covered during their respective breaking periods. These periods are identified as those where signal b matches some braking pattern, and are the intervals (r, t_1) for C_1 and (r, t_2) for C_2. Integrating v_i over respective intervals (r, t_i) for $i = 1..2$ we get a numerical measure and conclude that C_1 performs better.

In this paper we propose a declarative and formal *measure specification language* for automatically extracting measures from hybrid discrete-continuous traces. The patterns that define the scope of measurements are expressed using a variant of the *timed regular expressions* (TRE) of [3, 2], specially adapted for this purpose by adding preconditions, postconditions and events. An additional language layer is used to define the particular measures applied to the matching segments. The actual extraction of the measures takes advantage of the recent pattern matching procedure introduced in [19] for computing the set of segments of a Boolean signal that match a timed regular expression. In the general case, the number of such matches can be uncountable and the procedure of [19] represents them as a finite union of zones. In our language, where pattern boundaries are punctual events, we obtain a finite number of matches.

The resulting framework provides a step toward making the common practice of quantitative measurement extraction more rigorous, bridging the gap between qualitative verification and quantitative performance evaluation. We demonstrate the applicability of our approach using the Distributed System Interface (DSI3) standard protocol [15] developed by the automotive industry. We formalize in our language measurements of some features described in the standard, extract them from simulation traces and report the performance of our prototype implementation.

Related Work

The approach proposed in this paper builds upon the timed regular expressions introduced in [3, 2] and shown there to be equivalent in expressive power to timed automata. We omit the renaming operator used for this expressivity theoretical result and enrich the formalism with other features that lead to a pattern language dedicated to measurements, which we call conditional TRE. Precondition and postcondition constraints allow us to express zero-duration events such as rising and falling edges of dense-time Boolean signals. Focusing on patterns that start and end with an event, the pattern matching algorithm of [19] returns a finite number of matching segments.

Our approach differs in several respects from monitoring procedures based on real-time temporal logics and their extensions to real-valued signals such as STL [16]. In a nutshell here is the difference between satisfaction in temporal logic and matching in regular expression. For any temporal logic with future operators, satisfaction of φ by a behavior w is defined as $(w, 0) \models \varphi$. To compute this satisfaction value of φ at 0 we need to compute $(w, t) \models \psi$ for some subformulas ψ of φ and some time $t \geq 0$, in other words determine whether some *suffix* of w satisfies ψ. This can be achieved by associating with every formula φ a satisfaction signal relative to w which is true for every t such that $(w, t) \models \varphi$. On the other hand, the matching of a regular expression φ in w is not defined relative to a single time point but to a *pair* of points (t, t') such that the segment of w between t and t' satisfies the expression. This property of regular expressions makes them ideal for defining intervals that match patterns.

The recent work on assertion-based features [7] is similar in spirit to ours. The authors propose an approach for quantitative evaluation of mixed-signal design properties expressed as regular expressions. In contrast to our work, the regular expressions are extended with local variables, which are used to explicitly store values of interest, such as the beginning and the end time of a matched pattern. This work addresses the problem of measuring properties (features) of hybrid automata models using formal methods. We also mention the extension to TRE proposed in [13] that combines specification of real-time events and states occurring in continuous-time signals. Their syntax and primitive constructs are inspired by and extend industrial standards PSL [10] and SVA [20]. This work focuses on a translation from TRE to timed automata acceptors, but does not address the problem of pattern matching an expression on a concrete trace.

In the context of modeling resource-constrained computations, quantitative languages [6] were studied as generalizations of formal languages in which traces are associated with a real number rather than a Boolean value. The authors use weighted automata to define several classes of quantitative languages and determine the trace values by computing maximum, limsup, liminf, limit average and discounted sum over a (possibly infinite) trace. The ideas of quantitative languages are further extended in [14], by defining the *model measuring* problem. The model checking problems of TCTL and LTL are extended in [21, 11, 1] to a model measuring paradigm by parameterizing the bounds of the temporal operators. The authors propose algorithms for identifying minimum and maximum parameter values for which the model satisfies the temporal formula. A similar extension is proposed in [4] for signal temporal logic (STL), where both the temporal bounds and real-valued thresholds are written as parameters and inferred from signals. Robust interpretation of temporal logic specifications [12, 9, 8] is another way to associate numbers with traces according to how strongly they satisfy or violate a property.

Hardware designers and others who use block diagrams for control and signal processing often realize measurement using additional observer blocks, but these are restricted to online measurements. As a result commercial circuit simulation suites offer scripting languages or built-in functions dedicated to measurement extraction, such as the `.measure` (Synopsys) and `.extract` (Mentor Graphics) libraries. The former is structured according to the notion of *trigger* and *target* events, the measurement being performed on the segment(s) of the trace in between. This is particularly suited for timing analysis such as rise-time or propagation time. The latter is more general but relies mostly on functional composition. Absolute time of events in the trace can be found by threshold crossing functions, and then passed on as parameters to other measurement primitives to apply an aggregating function over suitable time intervals. In the approach we propose, one gains the expressiveness of the language of timed regular expression, that allow to detect complex sequences of events and states in the trace. This facilitates repeated measurements over a sequence of specified patterns, by clearly separating the behavior description from the measure itself.

2 Timed Regular Expression Patterns

In this section, we first recall the definition of the *timed regular expressions* (TRE) from [19]. Such expressions were defined over Boolean signals and in order to use them for real-valued signals we add predicates on real values to derive Boolean signals. This straightforward extension is still not entirely suitable for defining measurement segments, for the simple reason that an arbitrary regular expression may have infinitely many matches. For example an atomic proposition p is matched by all sub-segments of a dense-time Boolean signal where p continuously holds. Consequently in the second part of this section, we propose a novel extension that we call *conditional* timed regular expressions (CTRE). This extension enables to condition the match of a TRE to a prefix and suffix, and allows defining *events* of zero duration. We define a restriction to CTRE, that we call *event-bounded* timed regular expressions (E-TRE), which guarantees that the set of patterns matching a E-TRE is always finite. Thanks to this finiteness property, we will use E-TRE as the main building block in defining our measurement specification language.

2.1 Timed Regular Expressions

Let X and B be sets of *real* and *propositional* variables and $w : [0, d] \to \mathbb{R}^m \times \mathbb{B}^n$, where $m = |X|$ and $n = |B|$, a multi-dimensional signal of length d. For a variable $v \in X \cup B$ we denote by $\pi_v(w)$ the projection of w on its component v.
A propositional variable $b \in B$ admits a negation $\neg b$, which value at time t is the opposite of that of b. For θ a concrete predicate $\mathbb{R} \to \mathbb{B}$ we may create a propositional symbol $\theta(x)$ which interpretation at time t will be given by the evaluation of θ on the value of real variable x at time t. We define the projection of w on $\neg b$ by letting $\pi_{\neg b}(w)[t] = 1 - \pi_b(w)[t]$, and the projection of w on $\theta(x)$ by letting $\pi_{\theta(x)}(w)[t] = \theta(\pi_x(w)[t])$. A *proposition* p is taken to be either a variable $b \in B$, a predicate $\theta(x)$ over some real variable x, or their negation $\neg b$ and $\neg\theta(x)$ respectively. We assume a given set of real predicates and take P the set of propositions derived from real and propositional variables as described. A signal is said to have *finite variability* if for every proposition $p \in P$ the set of discontinuities of $\pi_p(w)$ is finite.

We now define the syntax of *timed regular expressions* according to the following grammar:

$$\varphi := \epsilon \mid p \mid \varphi_1 \cdot \varphi_2 \mid \varphi_1 \cup \varphi_2 \mid \varphi_1 \cap \varphi_2 \mid \varphi^* \mid \langle\varphi\rangle_I$$

where p is a proposition of P, and I is an interval of $\mathbb{R}_+$.

The semantics of a timed regular expression φ with respect to a signal w and times $t \leq t'$ in $[0, d]$ is given in terms of a satisfaction relation $(w, t, t') \models \varphi$

inductively defined as follows:

$$\begin{array}{lll}
(w,t,t') \models \epsilon & \leftrightarrow & t = t' \\
(w,t,t') \models p & \leftrightarrow & t < t' \text{ and } \forall\, t < t'' < t',\ \pi_p(w)[t''] = 1 \\
(w,t,t') \models \varphi_1 \cdot \varphi_2 & \leftrightarrow & \exists\, t \leq t'' \leq t',\ (w,t,t'') \models \varphi_1 \text{ and } (w,t'',t') \models \varphi_2 \\
(w,t,t') \models \varphi_1 \cup \varphi_2 & \leftrightarrow & (w,t,t') \models \varphi_1 \text{ or } (w,t,t') \models \varphi_2 \\
(w,t,t') \models \varphi_1 \cap \varphi_2 & \leftrightarrow & (w,t,t') \models \varphi_1 \text{ and } (w,t,t') \models \varphi_2 \\
(w,t,t') \models \varphi^* & \leftrightarrow & (w,t,t') \models \epsilon \text{ or } (w,t,t') \models \varphi \cdot \varphi^* \\
(w,t,t') \models \langle \varphi \rangle_I & \leftrightarrow & t' - t \in I \text{ and } (w,t,t') \models \varphi
\end{array}$$

Following the definitions in [19], we characterize the set of segments of w that match an expression φ by their *match set*. The match set of expression φ over w is the set of all pairs (t,t') such that the segment of w between t and t' matches φ.

Definition 1 (Match Set). *For any signal w and expression φ, we define their match set as*

$$\mathcal{M}(\varphi, w) := \{(t,t') \in \mathbb{R}^2 \mid (w,t,t') \models \varphi\}$$

We recall that a match set is a subset of $[0,d] \times [0,d]$ confined to the upper triangle defined by $t \leq t'$ taking t, t' the first and second coordinates of $\mathbb{R}^2$. It has been established that such a set can always be represented as a finite union of *zones*. In $\mathbb{R}^n$, zones are a special class of convex polytopes definable by intersections of inequalities of the form $x_i \geq a_i$, $x_i \leq b_i$ and $x_i - x_j \leq c_{i,j}$ or corresponding strict inequalities. We say that a zone is *punctual* when the value of each variable is uniquely defined, with for instance $a_i = b_i$ for all $i = 1..n$. We use zones in $\mathbb{R}^2$ to describe the relation between t and t' in a match set.

Theorem 1 ([19]). *For any finite variability signal w and* TRE *φ, the set $\mathcal{M}(\varphi, w)$ is a finite union of zones.*

2.2 Conditional TRE

We propose in the sequel *conditional timed regular expressions* (CTRE) that extend TRE. This extension enables to condition the match of a TRE to a prefix or a suffix. We introduce in the syntax of CTRE two new binary operators, "?" for preconditions, and "!" for postconditions. For some expressions φ_1 and φ_2 a trace w matches the expression $\varphi_1 \,?\, \varphi_2$ at (t,t') if it matches φ_2 and there is an interval ending at t where w matches φ_1. Symmetrically w matches the expression $\varphi_1 \,!\, \varphi_2$ at (t,t') if it matches φ_1 and there is an interval beginning at t' where w matches φ_2. We define formally the semantics of these operators for φ_1, φ_2 arbitrary CTRE and w an arbitrary signal as follows:

$$\begin{array}{lll}
(w,t,t') \models \varphi_1 \,?\, \varphi_2 & \leftrightarrow & (w,t,t') \models \varphi_2 \text{ and } \exists t'' \leq t,\ (w,t'',t) \models \varphi_1 \\
(w,t,t') \models \varphi_1 \,!\, \varphi_2 & \leftrightarrow & (w,t,t') \models \varphi_1 \text{ and } \exists t'' \geq t',\ (w,t',t'') \models \varphi_2
\end{array}$$

A precondition φ_1 and a postcondition φ_3 can be associated to an expression φ_2 independently as we have $\varphi_1 \,?(\varphi_2 \,!\, \varphi_3) \equiv (\varphi_1 \,?\, \varphi_2) \,!\, \varphi_3$ so that such expressions

may be noted $\varphi_1 \,?\, \varphi_2 \,!\, \varphi_3$ without ambiguity. Associating several conditions can form a sequential condition as with $(\varphi_1 \,?\, \varphi_2) \,?\, \varphi_3 \equiv (\varphi_1 \cdot \varphi_2) \,?\, \varphi_3$, or conjoint conditions as with $\varphi_1 \,?(\varphi_2 \,?\, \varphi_3) \equiv (\varphi_1 \,?\, \varphi_3) \cap (\varphi_2 \,?\, \varphi_3)$. There are further relationships with respect to other TRE operators, which we will not detail.

2.3 TRE with Events

Another important aspect of CTRE is that they enable defining *rise* and *fall events* of zero duration associated to propositional terms. The rise edge $\uparrow p$ associated to the propositional term p is obtained by syntactic sugar as $\uparrow p := \neg p \,?\, \epsilon \,!\, p$, while the fall edge $\downarrow p$ corresponds to $\downarrow p := \uparrow \neg p$. We now define a restriction of CTRE that we call TRE *with events*. This sub-class of CTRE consists of restricting the use of conditional operators to the definition of events. The introduction of events in TRE still guarantees the finite representation of their match set.

Corollary 1 (of Theorem 1). *For any finite variability signal w and* TRE *with events φ, the set $\mathcal{M}(\varphi, w)$ is a finite union of zones.*

Proof. By induction on the expression structure. For expressions of the form $\varphi = \uparrow p$, the match set $\mathcal{M}(\varphi, w)$ is of the form $\{(t, t) \;:\; t \in R\}$. By finite variability hypothesis R is finite as contained in the set of discontinuities of p, and in particular $\mathcal{M}(\varphi, w)$ is a finite union of punctual zones. All other operators are part of the grammar of timed regular expressions, and the proof of Theorem 1 grants us the property.

In what follows we consider events to be part of the syntax of timed regular expressions, and will just write TRE instead of TRE *with events*.

Remark Our support for events is minimal as compared to the *real-time regular expressions* of [13] where the authors use special operators `##0` and `##1` for event concatenation. Their work extends discrete-time specification languages, which have the supplementary notion of *clocks* noted $\texttt{@}(\uparrow c)$ with c a Boolean variable, and the implicit notion of *clock context*. A clock $\texttt{@}(\uparrow c)$ can then be used in conjunction with a proposition p to form a clocked event noted $\texttt{@}(\uparrow c)\, p$. Such an event allows to probe the value of p at the exact times where $\uparrow c$ occurs, which we did not consider. Assuming we dispose of atomic expressions $\texttt{@}(\uparrow c)\, p$ holding punctually at times such that $\uparrow c$ occurs and p is true, the event concatenation `##1` can be emulated by $\texttt{@}(\uparrow c)\, p \;\texttt{##1}\; \texttt{@}(\uparrow d)\, q \equiv \texttt{@}(\uparrow c)\, p \cdot d^* \cdot \neg d \cdot \texttt{@}(\uparrow d)\, q$.

We now say that a TRE is *event-bounded* when of the form $\uparrow p$, $\psi_1 \cdot \varphi \cdot \psi_2$, $\psi_1 \cup \psi_2$, or $\psi_1 \cap \varphi$ with p a proposition, and ψ_1, ψ_2 event-bounded TRE. Such expressions, that we call E-TRE for short, have an important "well-behaving" property as follows. Given an arbitrary finitely variable signal w, an E-TRE can be matched in w only a finite number of times. In the following lemma, we demonstrate that the match set for arbitrary finite signal w and E-TRE ψ consists of a finite number of points (t, t') with t an occurrence of a begin event and t' an occurrence of an end event.

Lemma 1. *Given an* E-TRE ψ *and a signal* w*, their associated match set* $\mathcal{M}(\psi, w)$ *is* finite*.*

Proof. By induction on the expression structure. Consider an arbitrary signal w and an event $\uparrow p$; by finite variability assumption there are finitely many time points in w where $\uparrow p$ occurs, so that its match set relatively to w is finite. Now let ψ be an E-TRE of the form $\psi = \psi_1 \cdot \varphi \cdot \psi_2$. The signal w matches ψ on the segment (t, t') if and only if there exists some times s and s' such that w matches ψ_1 on (t, s) and matches ψ_2 on (s', t'). By induction hypothesis there are finitely many such times t, t', s and s' so that ψ itself has a finite number of matches. One can easily see that the finiteness of the match set is also preserved by unions and intersections $\psi_1 \cup \psi_2$ and $\psi_1 \cap \varphi$, which concludes our proof.

3 Measuring with Conditional TRE

In this section, we propose a language for describing mixed-signal measures, and a procedure to compute such measures. In our approach, we will use *measure patterns* based on timed regular expressions to specify signal segments of interest. More precisely, a measure pattern consists of three parts: (1) the *main* pattern; (2) the *precondition*; and (3) the *postcondition*. The main pattern is an E-TRE that specifies the portion of the signal over which the measure is taken. Using E-TRE to express main patterns ensures the finiteness of signal segments, while pre- and post- conditions expressed as general TRE allow to define additional constraints. We formally define measure patterns as follows.

Definition 2 (Measure Pattern). *A measure pattern* φ *is a* CTRE *of the form* $\alpha \,?\, \psi \,!\, \beta$*, where* α *and* β *are* TRE*, while* ψ *is an* E-TRE

Note that preconditions and postconditions can be made optional by using ϵ as we have $\epsilon \,?\, \varphi \equiv \varphi$ and $\varphi \,!\, \epsilon \equiv \varphi$. In what follows we may use simpler formulas to express their semantic equivalent, for instance writing φ to refer to the measure pattern $\epsilon \,?\, \varphi \,!\, \epsilon$.

According to previous definitions, the match set of a measure pattern $\alpha \,?\, \psi \,!\, \beta$ gives us the set of all segments of the signal, represented as couples (t, t'), such that $(w, t, t') \models \psi$, and w satisfies both the precondition α before t and the postcondition β after t'.

Proposition 1. *For any signal* w *and a pattern* $\varphi = \alpha \,?\, \psi \,!\, \beta$*, their associated match set set is given by*

$$\mathcal{M}(\varphi, w) = \{(t, t') \,:\, \exists s \leq t \leq t' \leq s',\, (w, s, t) \models \alpha \text{ and } (w, t, t') \models \psi \text{ and } (w, t', s') \models \beta \,\}$$

Theorem 2 (Match set Finiteness). *For any signal* w *and measure pattern* $\varphi = \alpha \,?\, \psi \,!\, \beta$*, their associated match set* $\mathcal{M}(\varphi, w)$ *is* finite*.*

Proof. This is a direct consequence of Lemma 1. The set $\mathcal{M}(\varphi, w)$ is included in $\mathcal{M}(\psi, w)$, which makes it finite.

The match set of a measure pattern may be obtained by selecting the punctual zones of $\mathcal{M}(\psi, w)$ that meet a zone of $\mathcal{M}(\alpha, w)$ at the beginning, and a zone of $\mathcal{M}(\beta, w)$ at the end. Match sets of arbitrary TRE are computable following the proof of Theorem 1. The overall procedure to compute the match set of a measure pattern appears as Algorithm 1. It uses the procedure ZONES(φ, w) as appearing in [19] which returns a set of zones whose union is equal to $\mathcal{M}(\varphi, w)$ for any timed regular expression φ and signal w. For a zone z we denote by $\pi_1(z)$ and $\pi_2(z)$ projections on its first and second coordinates respectively.

Algorithm 1 Computation of the match set $\mathcal{M}(\varphi, w)$.

Require: measure pattern $\varphi = \alpha\,?\,\psi\,!\,\beta$, signal w
Ensure: $\mathcal{M}(\varphi, w)$
1: $\mathcal{M}(\varphi, w) \leftarrow \emptyset$
2: $Z_\alpha \leftarrow$ ZONES(α, w)
3: $Z_\beta \leftarrow$ ZONES(β, w)
4: $Z_\psi \leftarrow$ ZONES(ψ, w)
5: **for all** $\{(t, t')\} \in Z_\psi$ **do**
6: **for all** $z \in Z_\alpha, z' \in Z_\beta$ **do**
7: **if** $t \in \pi_2(z)$ and $t' \in \pi_1(z')$ **then**
8: $\mathcal{M}(\varphi, w) \leftarrow \mathcal{M}(\varphi, w) \cup \{(t, t')\}$
9: **end if**
10: **end for**
11: **end for**
12: **return** $\mathcal{M}(\varphi, w)$

The computation of a match set for a measure pattern φ and a signal w enables powerful pattern-driven performance evaluation of hybrid or continuous systems. Once the associated match set $\mathcal{M}(\varphi, w)$ is computed, we propose a two stage analysis of signals.

In the first step, we compute a scalar value for each segment of w that matches φ, either from absolute times of that match, or from the values of a real signal x in w during that match. A measure is then written with the syntax $\mathrm{op}(\varphi)$ with $\mathrm{op} \in \{\mathsf{time}, \mathsf{value}_x, \mathsf{duration}, \mathsf{inf}_x, \mathsf{sup}_x, \mathsf{integral}_x, \mathsf{average}_x\}$ being some sampling or aggregating operator. The semantics $[\![\,]\!]_w$ of these operators is given in Table 1; it associates to a measure $\mathrm{op}(\varphi)$ and trace w a multiset containing the scalar values computed over each matched interval.[1]

In the second step, we reduce the multiset of scalar values computed over the signal matched intervals in $\mathcal{M}(\varphi, w)$ to a single scalar. Typically, given the multiset $A = [\![\mathrm{op}(\varphi)]\!]_w$ of scalar values associated with these signal segments, this

[1] We use *multiset* semantics as several patterns may have exactly the same measured value, in which case *set* semantics would not record its number of occurrences.

Table 1. Standard measure operators.

$$
\begin{array}{ll}
[\![\mathsf{time}(\uparrow p)]\!]_w & = \{t \,:\, (t,t) \in \mathcal{M}(\uparrow p, w)\} \\
[\![\mathsf{value}_x(\uparrow p)]\!]_w & = \{\pi_x(w)[t] \,:\, (t,t) \in \mathcal{M}(\uparrow p, w)\} \\
[\![\mathsf{duration}(\varphi)]\!]_w & = \{t' - t \,:\, (t,t') \in \mathcal{M}(\varphi, w)\} \\
[\![\mathsf{inf}_x(\varphi)]\!]_w & = \{\min_{t \le \tau \le t'} \pi_x(w)(\tau) \,:\, (t,t') \in \mathcal{M}(\varphi, w)\} \\
[\![\mathsf{sup}_x(\varphi)]\!]_w & = \{\max_{t \le \tau \le t'} \pi_x(w)(\tau) \,:\, (t,t') \in \mathcal{M}(\varphi, w)\} \\
[\![\mathsf{integral}_x(\varphi)]\!]_w & = \{\int_t^{t'} \pi_x(w)(\tau) d\tau \,:\, (t,t') \in \mathcal{M}(\varphi, w)\} \\
[\![\mathsf{average}_x(\varphi)]\!]_w & = \{\frac{1}{t'-t} \int_t^{t'} \pi_x(w)(\tau) d\tau \,:\, (t,t') \in \mathcal{M}(\varphi, w)\}
\end{array}
$$

phase consists in computing standard statistical indicators over A, such as the average, maximum, minimum or standard deviation. This final step is optional, the set of basic measurements sometimes provides sufficient information.

Anti-lock Brake System Example We now refer back to our first example from Figure 1 and propose measure pattern formalization to evaluate performance of the controller. We first formalize the pattern of a brake control signal b under a (heavy) braking situation. The main pattern ψ starts with a rise event on b and a braking period with the duration in I, continues with one or more pulses with duration in J, and ends with a fall event on b:

$$\psi := \uparrow b \cdot \langle b \rangle_I \cdot \langle \neg b \cdot b \rangle_J^+ \cdot \downarrow b$$

We also need to ensure that the speed should be zero at the end of braking situation, with the postcondition $\beta := (v \le 0)$. Finally, we can measure the stopping distance using the expression

$$\mathsf{integral}_v(\psi \,!\, \beta)$$

integrating v over intervals matching the measure pattern.

4 Case Study

4.1 Distributed Systems Interface

Distributed systems interface (DSI3) is a flexible and powerful bus standard [15] developed by the automotive industry. It is designed to interconnect multiple remote sensor and actuator devices to a controller. The controller interacts with the sensor devices via so-called *voltage* and *current lines*. In this paper we focus on two phases of the DSI3 protocol:

- the initialization phase called the *discovery mode*;
- one of the stationary phases called the *command and response mode*.

In the discovery mode, prior to any interaction the power is turned on, resulting in a voltage ramp from 0V to V_{high}. The communication is initiated by the controller that probes the presence/absence of sensors by emitting analog pulses on the voltage line. Connected sensor devices respond in turn with another pulse sent over the current line. At the end of this interaction, a final short pulse is sent to the sensors interfaces, marking the end of the discovery mode.
In the command and response mode, the controller sends a command to the sensor as a series of pulses (or pulse train) on the voltage line, which transmits its response by another pulse train on the current line. For power-demanding applications the command-response pairs are followed by a power pulse, which goes above V_{high}. This allows the sensor to load a capacitor used for powering its internal operation.

The DSI3 standard provides a number of ordering and timing requirements that determine correct communication between the controller and the sensor devices: (1) minimal time between the power turned on and first discovery pulse; (2) maximal duration of discovery mode; (3) expected time between two consecutive discovery pulses; (4) expected time between command and response. Figure 2 illustrates the discovery mode in the DSI3 protocol and provides a high-level overview of its ordering and timing requirements. In this example, the controller probes five sensor interfaces.

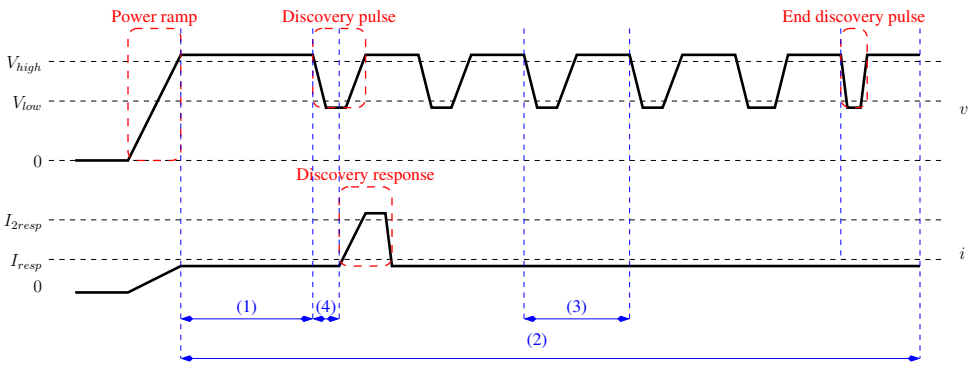


Fig. 2. DSI3 discovery mode – overview.

The correctness of a DSI3 protocol implementation in an automotive airbag system was studied in [17]. The above requirements were formalized as assertions expressed in *signal temporal logic* (STL) and the monitoring tool AMT [18] was used to evaluate the simulation traces. In this paper we do more than checking correctness, but evaluate the performance of a controller and sensor implementation. We use measure patterns to specify signal segments of interest and define several measures within the framework introduced in Section 3. We study two specific measures: (1) the time between consecutive discovery pulses; and (2) the amount of energy transmitted to the sensor through power pulses.

In order to generate simulation traces, we model our system as follows: the controller is a voltage-source, and the sensor is a current-source in parallel with a resistive-capacitive load. The schematic is shown in Figure 3. During the discovery phase the load is disabled; the voltage source generates randomized pulses in which the time between two discovery pulses has a Gaussian distribution with a mean of 250μs and a standard deviation of 3.65μs. During the power pulses of the command and response mode, the load is enabled and randomized, with $C = 120$nF and R uniformly distributed in the range $[25\Omega, 35\Omega]$. Threshold levels are 4.6V low, 7.8V high, 8.3V power, and 11.5V idle.

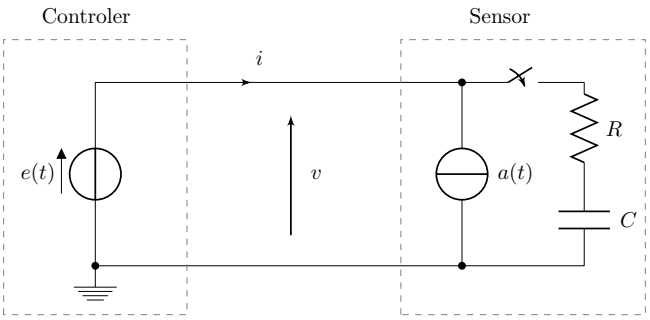


Fig. 3. Electrical model of the system.

4.2 Measurements

Time between consecutive discovery pulses In order to characterize a discovery pulse, we first define three regions of interest – when the voltage v is (1) below V_{low}; (2) between V_{low} and V_{high}; and (3) above V_{high}. We specify these regions with the following predicates:

$$\begin{aligned} v_l &\equiv v \leq V_{low} \\ v_b &\equiv V_{low} \leq v \leq V_{high} \\ v_h &\equiv v \geq V_{high} \end{aligned}$$

Next, we describe the shape of a discovery pulse. Such a pulse starts at the moment when the signal v moves from v_h to v_b. The signal then must go into v_l, v_b and finally come back to v_h. In addition to its shape, the DSI3 specification requires the discovery pulse to have a certain duration between some d_{min} and d_{max}. This timing requirement allows distinguishing a discovery pulse from other pulses, such as the end-of-discovery pulse. We illustrate the requirements for a discovery pulse in Figure 4-a and formalize it with the following E-TRE:

$$\psi_{dp} \equiv \downarrow (v_h) \cdot \langle v_b \cdot v_l \cdot v_b \rangle_{[d_{min}, d_{max}]} \cdot \uparrow (v_h)$$

In order to measure the time between consecutive discovery pulses, we need to characterize signal segments that we want to measure. The associated pattern shall start at the beginning of a discovery pulse and end at the beginning of the next one, as depicted by the ψ region in Figure 4-a. It consists of a discovery pulse ψ_{dp}, followed by the voltage signal being in the v_h region, and terminating when the voltage leaves v_h. This description is not sufficient – we also need to ensure that this segment is effectively followed by another discovery pulse. Hence we add a postcondition that specifies this additional constraint. The measure pattern $\varphi_1 \equiv \alpha_1 \,?\, \psi_1 \,!\, \beta_1$ is formalized as follows.

$$\begin{aligned} \alpha_1 &\equiv \epsilon \\ \psi_1 &\equiv \psi_{dp} \cdot v_h \cdot \downarrow (v_h) \\ \beta_1 &\equiv \psi_{dp} \end{aligned}$$

Finally, we evaluate the measure expression $\mu_1 := \mathsf{duration}(\varphi_1)$ over signal w.

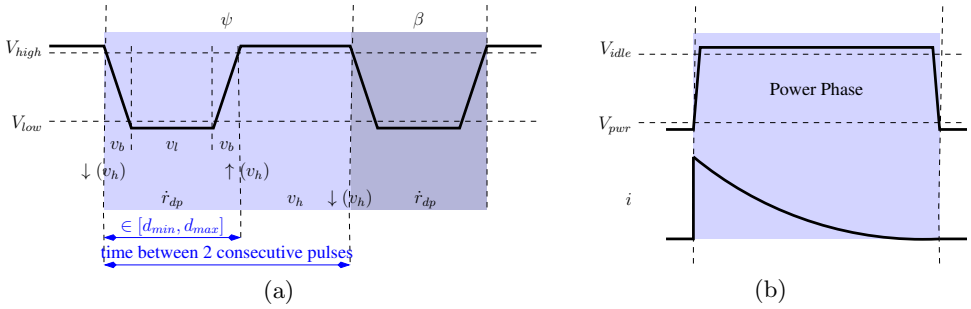


Fig. 4. (a) Consecutive discovery pulses with timing; (b) Power pulse and flow.

Energy transfer from controller to sensor In the DSI3 protocol, the discovery mode can be followed by a stationary command and respond mode. A command and respond mode sequence is a pulse train that consists of a command subsequence in the form of potential pulses between V_{high} and V_{low}, a response subsequence by means of current pulses between 0 and I_{resp}, and finishes by a power pulse rising to potential V_{idle} in which a large current can be drawn by the sensor. We first characterize the power pulse as depicted in Figure 4-b. It occurs when the voltage goes from below V_{pwr} to above V_{idle}, and back under V_{pwr}. The three regions of interest are specified with the following predicates.

$$\begin{aligned} v_h &\equiv v \geq V_{pwr} \\ v_t &\equiv V_{pwr} \leq v \leq V_{idle} \\ v_p &\equiv v \geq V_{idle} \end{aligned}$$

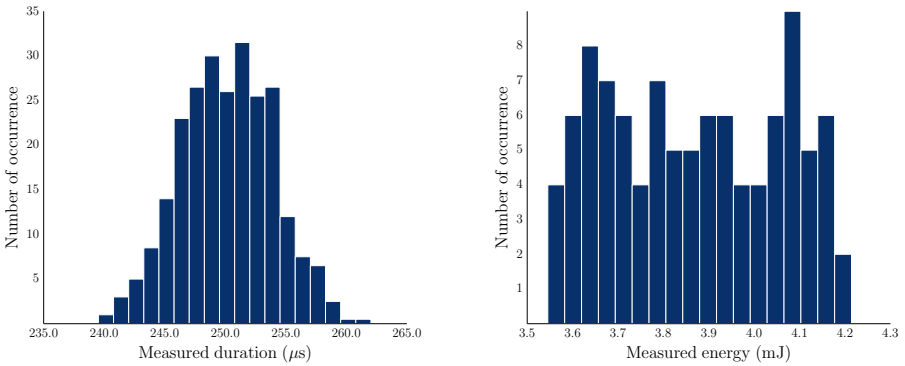
Hence the pattern specifying a power pulse is expressed as

$$\psi_2 \equiv \uparrow (v_h) \cdot v_t \cdot v_p \cdot v_t \cdot \downarrow (v_h)$$

The measure pattern does not have pre- or post-conditions as all other communications occur with v below V_{idle}, hence $\alpha_2 = \beta_2 = \epsilon$. The measure pattern φ_2 is equivalent to its main pattern ψ_2. Given v the voltage and i the current on the communication line, the energy transfered to the sensor is given by the area under the signal $v \times i$ between the start and end of power pulse. We assume that such a signal is given in the simulation trace w, and evaluate the measure expression $\mu_2 := \mathsf{integral}_{v \times i}(\psi_2)$ over signal w.

4.3 Experimental Results

We extended the prototype tool developed in [19] with algorithms for matching zero-duration events and conditional TRE as appearing in measure patterns, and with the support of measure operations introduced in Section 3. The implementation was done in Python and uses the C library from IF [5] for computing operations on zones. For our experiment we apply a scenario according to which our electrical model is switched on/off 100 times in sequence to stress the discovery mode of DSI3. The set of traces we generate conform to the discovery, and command-and-response modes of the protocol. We then compute match sets for properties presented in Section 4.2 over these simulation traces using our prototype implementation. In Figure 5, we depict measurement results using histograms. The distribution of the times between two discovery pulses follows a normal distribution according to the timing parameters used to generate it. The energy transfered to the sensor through power pulses has a flatter distribution as the result of a uniformly distributed load resistance value.



(a) (b)

Fig. 5. (a) Distribution of μ_1, the time between two consecutive discovery pulses; (b) Distribution of μ_2, the energy transmitted per power pulse.

We then compared the execution times to compute measurements, using a periodic sampling with different sampling rates – note that our method supports variable step sampling without extra cost. The computation times are given in Table 2 with the detailed computation time needed for predicate evaluation (T_p), match set computation (T_m), measure aggregation (T_a) and total computation time (T). Computation of match sets does not depend on the number of samples but on the number of uniform intervals of atomic propositions; evaluation of real predicates by linear interpolation, and computing measures like integration can be done in time linear in the number of samples.

Table 2. Computation times (s)

	Measure μ_1				Measure μ_2			
# samples	T_p	T_m	T_a	T	T_p	T_m	T_a	T
1M	0.047	0.617	0.000	**0.664**	0.009	0.004	0.011	**0.024**
5M	0.197	0.612	0.000	**0.809**	0.050	0.005	0.047	**0.103**
10M	0.386	0.606	0.000	**0.992**	0.101	0.005	0.100	**0.216**
20M	0.759	0.609	0.000	**1.368**	0.203	0.005	0.260	**0.468**

5 Conclusion and Future Work

We presented a formal measurement specification language that can be used for evaluating cyber-physical systems based on their simulation traces. Starting from a declarative specification of the patterns that should be matched in the segments to be measured, we apply a pattern matching algorithm for timed regular expressions to find out the scope of measurements. The applicability of our framework was demonstrated on a standard mixed-signal communication protocol from the automotive domain.

In the future, we plan to develop an online extension of the presented pattern matching and measurement procedure. It will enable the application of measurements during the simulation process as well as performing measurements on real cyber-physical systems during their execution. We believe that the extension of regular expressions that we introduced is sufficiently expressive to capture common mixed signal properties, and could be used in other application domains, something that we intend to explore further.

Acknowledgements This work was supported by ANR project CADMIDIA, and the MISTRAL project A-1341-RT-GP coordinated by the European Defence Agency (EDA) and funded by 8 contributing Members (France, Germany, Italy, Poland, Austria, Sweden, Netherlands and Luxembourg) in the framework of the Joint Investment Programme on Second Innovative Concepts and Emerging Technologies (JIP-ICET 2).

References

1. Rajeev Alur, Kousha Etessami, Salvatore La Torre, and Doron Peled. Parametric temporal logic for "model measuring". *ACM Transactions on Computational Logic (TOCL)*, 2(3):388–407, 2001.
2. Eugene Asarin, Paul Caspi, and Oded Maler. A Kleene theorem for timed automata. In *Logic in Computer Science (LICS)*, pages 160–171, 1997.
3. Eugene Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *Journal of ACM*, 49(2):172–206, 2002.
4. Eugene Asarin, Alexandre Donzé, Oded Maler, and Dejan Nickovic. Parametric identification of temporal properties. In *Runtime Verification (RV)*, pages 147–160, 2011.
5. Marius Bozga, Susanne Graf, and Laurent Mounier. IF-2.0: A validation environment for component-based real-time systems. In *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, pages 343–348, 2002.
6. Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Transactions on Computational Logic (TOCL)*, 11(4):23, 2010.
7. Antonio Anastasio Bruto da Costa and Pallab Dasgupta. Formal interpretation of assertion-based features on AMS designs. *IEEE Design & Test*, 32(1):9–17, 2015.
8. Alexandre Donzé, Thomas Ferrère, and Oded Maler. Efficient robust monitoring for STL. In *Computer Aided Verification (CAV)*, pages 264–279, 2013.
9. Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *Formal Modeling and Analysis of Timed Systems (FORMATS)*, pages 92–106, 2010.
10. Cindy Eisner and Dana Fisman. *A practical introduction to PSL*. Springer, 2006.
11. E. Allen Emerson and Richard J. Trefler. Parametric quantitative temporal reasoning. In *Logic in Computer Science (LICS)*, pages 336–343, 1999.
12. Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42), 2009.
13. John Havlicek and Scott Little. Realtime regular expressions for analog and mixed-signal assertions. In *Formal Methods in Computer-Aided Design (FMCAD)*, pages 155–162, 2011.
14. Thomas A. Henzinger and Jan Otop. From model checking to model measuring. In *Conference on Concurrency Theory (CONCUR)*, pages 273–287, 2013.
15. Distributed System Interface. *DSI3 Bus Standard*. DSI Consortium.
16. Oded Maler and Dejan Nickovic. Monitoring properties of analog and mixed-signal circuits. *STTT*, 15(3):247–268, 2013.
17. Thang Nguyen and Dejan Nickovic. Assertion-based monitoring in practice - checking correctness of an automotive sensor interface. In *Formal Methods for Industrial Critical Systems (FMICS)*, pages 16–32, 2014.
18. Dejan Nickovic and Oded Maler. AMT: A property-based monitoring tool for analog systems. In *Formal Modeling and Analysis of Timed Systems (FORMATS)*, pages 304–319, 2007.
19. Dogan Ulus, Thomas Ferrère, Eugene Asarin, and Oded Maler. Timed pattern matching. In *Formal Modeling and Analysis of Timed Systems (FORMATS)*, pages 222–236, 2014.
20. Srikanth Vijayaraghavan and Meyyappan Ramanathan. *A practical guide for SystemVerilog assertions*. Springer, 2006.
21. Farn Wang. Parametric timing analysis for real-time systems. *Information and Computation*, 130(2):131–150, 1996.