Checking Temporal Properties of Discrete, Timed and Continuous Behaviors

Oded Maler[1], Dejan Nickovic[1], and Amir Pnueli[2,3]

[1] Verimag, 2 Av. de Vignate, 38610 Gières, France
Dejan.Nickovic@imag.fr, Oded.Maler@imag.fr
[2] Weizmann Institute of Science, Rehovot 76100, Israel
[3] New York University, 251 Mercer St. New York, NY 10012, USA
Amir.Pnueli@cs.nyu.edu

Words, even infinite words, have their limits.
Dedicated to B.A. Trakhtenbrot on his 85th Birthday.

Abstract. We survey some of the problems associated with checking whether a *given* behavior (a sequence, a Boolean signal or a continuous signal) satisfies a property specified in an appropriate temporal logic and describe two such monitoring algorithms for the real-time logic MITL.

1 Introduction

This paper is concerned with the following problem:

> *Given a temporal property φ how does one check that a given behavior ξ satisfies it.*

Within this paper we assume that the behavior to be checked is produced by a *model* of a dynamical system S, although some of the techniques are applicable to behaviors generated by real physical systems. Unlike formal verification which aims at showing that *all* behaviors generated by S satisfy φ, here S is used to generate *one behavior at a time* and can thus be viewed as a *black box*. This setting has been studied extensively in recent years both in the context of digital hardware, under the names of "dynamic" verification, or assertion checking as well as for software, where it is referred to as *runtime verification* [15,39]. We will use the term *monitoring*. In this framework the question of *coverage*, that is, finding a finite number of test cases whose behavior will guarantee overall correctness, is delegated outside the scope of the property monitor. This approach can be used when the system model is too large to be verified formally. It is also applicable when the "model" in question is nothing but a hardly-formalizable simulation program, as is often the case in electrical simulation of circuits. On the other hand, the explicit presentation of ξ itself, rather than using the generating model S, raises new problems.

Most of the work described in this paper has been performed within the European project PROSYD[1] with the purpose of extending some ingredients

[1] IST-2003-507219 PROSYD (Property-Based System Design).

A. Avron et al. (Eds.): Trakhtenbrot/Festschrift, LNCS 4800, pp. 475–505, 2008.

of verification methodology from digital (discrete) to analog (continuous and hybrid) systems. Consequently, we treat systems and behaviors described at three different levels of abstraction (discrete, timed and continuous). Hence we find it useful to start with a generic model of a dynamical system defined over an abstract state space which evolves in an abstract time domain, see also [28,29]. The three models used in the paper are obtained as special instances of this model.

States and Behaviors. A model S of a system is defined over a set $V = \{x_1, \ldots x_n\}$ of *state variables*, each ranging over a domain X_i. The *state space* of the system is thus $X = X_1 \times \cdots \times X_n$. The system evolves over a time domain T which is a linearly-ordered set. A *behavior* of the system is a function from the time domain to the state space, $\xi : T \to X$. We consider *complete* behaviors, where ξ is defined all over T, as well as *partial* behaviors where ξ is defined only on a downward-closed subset of T, that is, some interval of the form $[0, r)$. We use the notation $\xi[t_1, t_2]$ for the restriction of ξ to the interval $[t_1, t_2]$ and let $\xi[t] = \perp$ when $t \geq r$. We denote the set of all possible (complete and partial) behaviors over a set X by X^*.[2]

Systems. The dynamics of a system S is defined via a rule of the form $x' = f(x, u)$ which determines the future state x' as a function of the current state x and current input $u \in U$. As mentioned earlier, we do not have access to f and our interaction with the model is restricted to stimulating it with an input $\nu \in U^*$ and then observing and checking the generated behavior $\xi \in X^*$.

Properties. Regardless of the formalism used to express it, a property φ defines a subset L_φ of X^*. A property monitor is a device or algorithm for deciding whether a given behavior ξ satisfies φ (denoted by $\xi \models \varphi$) or, equivalently, whether $\xi \in L_\varphi$.

The paper starts with properties of discrete (digital) systems, a well-studied and mature domain, where some of the problems associated with monitoring (non-causality of the specification formalism, satisfiability by finite traces, online vs. offline) are already manifested. We then move to *timed* discrete systems, whose behaviors can be viewed as *continuous-time Boolean signals*, which raise a lot of new issues such as sampling, event detection, variability bounds, etc. Most of the paper will investigate monitoring at this level of abstraction where we made some original contributions. Finally we move to continuous (analog) signals which, in addition to dense time, admit also *numerical real values*. Although for many types of properties (and in particular those expressible in our *signal temporal logic* [37,31]) checking continuous properties can be reduced to checking timed properties, there are further issues, such as approximation errors, raised by the continuous domain and by the manner in which signals are generated by numerical simulators.

[2] For discrete time behaviors, it is common to use X^* for finite behaviors and X^ω for infinite ones, but these distinctions are less meaningful when we come to continuous behaviors.

2 Discrete (Digital) Systems: Properties

Discrete models are used for modeling digital hardware (at gate level and above) as well as software. At this level of abstraction the set $\mathbb{N}$ of natural numbers is taken as the underlying time domain. In this case the difference between $\xi[t]$ and $\xi[t+1]$ reflects the changes in state variables that took place in the system within one clock cycle (hardware) or one program step (software).[3] The state space of digital systems is often viewed as the set $\mathbb{B}^n$ of Boolean n-bit vectors.[4] Behaviors are, hence, n-dimensional Boolean *sequences* generated by system models which are essentially finite automata (transition systems) which can be encoded in a variety of formalisms such as systems of Boolean equations with primed variables or unit delays, hardware description languages at various levels of abstraction, programming languages, etc.

Semantically speaking, a property is a subset of the set of all sequences (also known in computer science as a *formal language*) indicating the behaviors that we allow the system to have. Such subsets can be defined syntactically using a variety of formalisms such as logical formulae, regular expressions or automata that accept them. In this paper we focus on *temporal logic* [35,36] which can be viewed as a useful syntactic sugar for the first-order fragment of the monadic logic of order [40]. This section does not present new results but is rather a synthetic survey of the state-of-the-art which can serve as an entry point to the vast literature and which, we feel, is a pre-requisite for understanding the timed and continuous extensions.

2.1 Temporal Logic (Future)

The temporal logic of linear time (LTL) is perhaps the most popular property specification formalism. In a nutshell it is a language for specifying certain relationships between values of the state variables at *different time instants*, that is, at different positions in the sequence. For example, we may require that whenever $x_1 = 1$ at position t then $x_2 = 0$ at position $t + 3$. A property monitor is thus a device that observes sequences and checks whether they satisfy all such relationships. We repeat briefly some standard definitions concerning the syntax and semantics of LTL. By *semantics* we mean the rules according to which a sequence is declared as satisfying or violating a formula φ.
The syntax of LTL is given by the following grammar:

$$\varphi := p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \bigcirc \varphi \mid \varphi_1 \mathcal{U} \varphi_2,$$

where p belongs to a set $P = \{p_1, \ldots, p_n\}$ of propositions indicating values of the corresponding state variable. The basic temporal operators are *next* ($\bigcirc$),

[3] We mention here the existence and usefulness of *asynchronous* (*event triggered* rather than *time triggered*) systems and models, where the interpretation of a step is different.

[4] In software, as well as in high-level models of hardware, systems may include state variables ranging over larger domains such as bounded and unbounded numerical variables or dynamically-varying data structures such as queues and trees, but, at least in the hardware context, those can be encoded by bit vectors.

which specifies what should hold in the next step and *until* ($\mathcal{U}$), which requires φ_1 to hold until φ_2 becomes true, without bounding the temporal distance to this becoming. From these basic LTL operators one can derive other standard Boolean operators as well as temporal operators such as *eventually* ($\Diamond$) and *always* ($\Box$):

$$\Diamond\varphi = \mathrm{T}\,\mathcal{U}\varphi \quad \text{and} \quad \Box\varphi = \neg\Diamond\neg\varphi.$$

Models of LTL are *Boolean sequences* of the form $\xi : \mathbb{N} \to \mathbb{B}^n$. We also use p to denote the sequence obtained by projecting a sequence ξ on the dimension corresponding to p. The satisfaction relation $(\xi, t) \models \varphi$, indicating that sequence ξ satisfies φ starting from position t, is defined inductively as follows:

$$\begin{array}{ll}
(\xi,t) \models p & \leftrightarrow p[t] = 1 \\
(\xi,t) \models \neg\varphi & \leftrightarrow (\xi,t) \not\models \varphi \\
(\xi,t) \models \varphi_1 \vee \varphi_2 & \leftrightarrow (\xi,t) \models \varphi_1 \text{ or } (\xi,t) \models \varphi_2 \\
(\xi,t) \models \bigcirc\varphi & \leftrightarrow (\xi,t+1) \models \varphi \\
(\xi,t) \models \varphi_1 \mathcal{U} \varphi_2 & \leftrightarrow \exists t' \geq t\ (\xi,t') \models \varphi_2 \text{ and } \forall t'' \in [t,t'), (\xi,t'') \models \varphi_1 \\
 & \\
(\xi,t) \models \Diamond\varphi & \leftrightarrow \exists t' \geq t\ (\xi,t') \models \varphi \\
(\xi,t) \models \Box\varphi & \leftrightarrow \forall t' \geq t\ (\xi,t') \models \varphi
\end{array}$$

A sequence ξ satisfies φ, denoted by $\xi \models \varphi$, iff $(\xi, 0) \models \varphi$.

2.2 Temporal Logic (Past)

The past fragment of LTL is defined by a syntax similar to the future fragment where the *next* and *until* operators are replaced by *previously* ($\ominus$) and *since* ($\mathcal{S}$). As with future LTL, useful derived operators are *sometime in the past* $\Diamond\!\!\!-$ and *always in the past* $\Box\!\!\!-$ defined as

$$\Diamond\!\!\!-\,\varphi = \mathrm{T}\,\mathcal{S}\varphi \quad \text{and} \quad \Box\!\!\!-\,\varphi = \neg\Diamond\!\!\!-\,\neg\varphi$$

Their semantics is given by

$$\begin{array}{ll}
(\xi,t) \models \ominus\varphi & \leftrightarrow t = 0 \text{ or } (\xi,t-1) \models \varphi \\
(\xi,t) \models \varphi_1 \mathcal{S} \varphi_2 & \leftrightarrow \exists t' \in [0,t]\ (\xi,t') \models \varphi_2 \text{ and } \forall t'' \in (t',t], (\xi,t'') \models \varphi_1 \\
 & \\
(\xi,t) \models \Diamond\!\!\!-\,\varphi & \leftrightarrow \exists t' \in [0,t](\xi,t') \models \varphi \\
(\xi,t) \models \Box\!\!\!-\,\varphi & \leftrightarrow \forall t' \in [0,t]\ (\xi,t') \models \varphi
\end{array}$$

A *finite* sequence satisfies a past property φ if it satisfies it from the last position "backwards", that is, $\xi \models \varphi$ if $(\xi, |\xi|) \models \varphi$.

3 Discrete Systems: Checking Temporal Properties

We describe here the fundamental problems associated with checking temporal properties as well as the common approaches for tackling them. These are problems that exist already in the simplest model of Boolean sequences and are propagated, with additional complications to the timed and continuous domains.

3.1 Causality and Non-determinism

A major difficulty in checking properties expressed in future LTL is due to the *non-causal* definition of the satisfaction relation. To see what this means it might be helpful to look at the definition of LTL semantics as a procedure which is *recursive* on both the structure of φ and on the sequential structure of ξ. This procedure is called initially with φ and with $\xi[0]$ as arguments because we want to determine the satisfiability of φ from position zero. Then the semantic rules "call" the procedure recursively with sub formulae of φ and with further positions of ξ. In other words, the satisfiability of φ at time t may depend on the value of ξ at some *future* time instant $t' \geq t$. Even worse, some temporal operators refer to future time instants in a *quantified* manner, for example, requiring some p to hold in *all future time instants*. The satisfiability of such a property may sometime be determined only at infinity, that is, "after" we can be sure that no instance of $\neg p$ is observed.

Note that for past LTL, the recursion goes *backward* in time and the satisfaction of a past formula φ by a sequence ξ at position t is determined according to the values of ξ at the interval $[0, t]$ and in this sense, past LTL is causal. However it has been argued that the futuristic specification style is more natural for humans. The past fragment of LTL admits an immediate translation to deterministic automata and a simple monitoring procedure [16] based on this observation.

The "classical" theoretical scheme for using LTL in formal verification is based on translating a formula φ into a non-deterministic automaton over infinite sequences (an ω-automaton) $\mathcal{A}_\varphi$ that accepts exactly the sequences that satisfy it. The non determinism is needed to compensate for the non causality: the automaton has to "guess" at time t whether future observations at some $t' > t$ will render φ satisfied at t, and split the computation into two paths according to these predictions. A path that made a wrong prediction will be aborted later, either within a finite number of steps (if the guess is falsified by some observation) or via the ω-acceptance condition (if the falsification is due to non-occurrence of an event at infinity). Satisfiability of the formula can thus be determined by checking whether the ω-language accepted by $\mathcal{A}_\varphi$ is not empty. This reduces to checking the existence of an accepting cycle in $\mathcal{A}_\varphi$ which is reachable from an initial state. Verification is achieved by checking whether S may generate an infinite behavior rejected by $\mathcal{A}_\varphi$ (or accepted by $\mathcal{A}_{\neg\varphi}$). It should be noted that simplified procedures have been developed and implemented when the property in question belongs to a subclass of LTL, such as safety.

3.2 Evaluating Incomplete Behaviors

In monitoring we do not exploit the model S that generates the sequences, but rather observe sequences as they come. The major problem here, with respect to the standard semantics of LTL which is defined over *complete infinite sequences*,

is the impossibility to observe infinite sequences in finite time.[5] Hence, the extension of LTL semantics to *incomplete behaviors* is a major issue in monitoring.

After having observed a finite sequence ξ we can be in one of the following three basic situations with respect to a property φ:

1. All possible infinite completions of ξ satisfy φ. Such a situation may happen, for example, when φ is $\Diamond p$ and p occurs in ξ. In this case we say that ξ *positively determines* φ.
2. All possible infinite completions of ξ violate φ. For example when φ is $\Box\neg p$ and p occurs in ξ. In this case we say that ξ *negatively determines* φ.
3. Some possible completions of ξ do satisfy φ and some others violate it. For example, any sequence where p has not occurred has extensions that satisfy, as well as extensions that violate, formulae such as $\Diamond p$ or $\Box\neg p$. In this case we say that ξ is *undecided*.

It should be noted that the "undecided" category can be refined according to both methodological, quantitative, and logical considerations. One might want to distinguish, for example, between "not yet violated" (in the case of $\Box\neg p$) and "not yet satisfied" (in the case of $\Diamond p$). The quantitative aspects enter the picture as well because the longer we observe a sequence ξ free of p, the more we tend to believe in the satisfaction of $\Box\neg p$, although the doubt will always remain. On the other hand, the satisfaction of a formula like $\bigcirc^k p$, a shorthand for $\bigcirc(\bigcirc(\ldots\bigcirc p)\ldots)$, although undecided for sequences shorter than k, will be revealed in finite time. The most general type of answer concerning the satisfiability of φ by a finite sequence ξ would be to give exactly the set of completions of ξ that will make it satisfy φ, defined as

$$\xi\backslash\varphi = \{\xi' : \xi \cdot \xi' \models \varphi\}.$$

Positive and negative determination correspond, respectively, to the special cases where $\xi\backslash\varphi = X^*$ and $\xi\backslash\varphi = \emptyset$. This "residual" language can be computed syntactically as the left quotient ("derivative") of φ by ξ.

In certain situations we would like to give a decisive answer at the end of the sequence. In the case of positive and negative determination we can reply with a yes/no answer. More general rules for assigning semantics to every finite sequence have been proposed [27,11]. Let us consider some sub-classes of LTL formulae for which such a finitary semantics clearly makes sense. The simplest among those is bounded-LTL where the only temporal operator is *next* and where satisfiability of a formula φ at time 0 is always determined by the values of the sequence up to some $t \leq k$, with k being a constant depending on φ. Note that this class is

[5] To be more precise, there are some classes of infinite sequences such as the *ultimately-periodic* ones, that admit a finite representation and an easily-checkable satisfiability, however we work under the assumption that we do not have much control over the type of sequences provided by the simulator and hence we have to treat arbitrary finite sequences. It is worth noting that if S is input-deterministic then an ultimately-periodic input induces an ultimately-periodic behavior.

not as useless as it might seem: one can use "syntactic sugar" operators such as $\Box_{[0,r]}\varphi$ as shorthand for $\bigwedge_{i=0}^{r-1}(\bigcirc^i\varphi)$. The implication for monitoring is that every *sufficiently-long* sequence is determined with respect to such formulae (see also [25]).

The next class is the class of *safety* properties[6] where the only quantification of the time variable is *universal* as in $\Box\varphi$. It is not hard to see that ω-languages corresponding to such formulae consist of infinite words that *do not have a prefix* in some finitary language. While monitoring a finite sequence ξ relative to such a formula, we can be in either of the following two situations. Either such a prefix has been observed and hence any continuation of ξ will be rejected and ξ can be declared as violating, or no such prefix has been observed but nothing prevents its occurrence in the future and ξ is undecided. A similar and dual situation holds for eventually property such as $\Diamond\varphi$ that quantify existentially over time, and where an occurrence of a finitary prefix satisfying φ renders the sequence accepted.

With respect to these sub-classes one can adopt the following policy: interpret any quantification $Qt, Q \in \{\forall, \exists\}$ as $Qt \leq |\xi|$ and hence a safety that has not been violated during the lifetime of ξ is considered as satisfied, and an eventuality not fulfilled by that time is interpreted as violated. This principle may be extended to more complex formulae that involve nesting of temporal operators but in this case the interpretation seems less intuitive.

Let us remark that although models of *past* LTL are finite sequences, the problem of undecided sequences still exists. Consider for example the property $\boxminus p$. As soon as $\neg p$ is observed, we can say the the formula is negatively determined and need not wait for the rest of the sequence. On the other hand, as long as $\neg p$ has not been observed, although the prefix satisfies the property we cannot give conclusive results until the "official" end of the sequence, because $\neg p$ may always be observed in the next instant. Hence the treatment of past properties is not much different from future ones, except for the simpler construction of the corresponding automaton

Naturally many solutions have been proposed to this problem in the context of monitoring and runtime verification and we mention few. The work of [1] concerning the FoCs property checker of IBM, as well as those of [22] are restricted to safety (prefix-closed) or eventuality properties and report violation when it occurs. On the other hand, the approach of giving the residual language is proposed in [23] and [41] in the context of timed properties. A systematic study of the possible adaptation of LTL semantics to finite sequences ("truncated paths") is presented in [11]. This semantics has been adopted by the semiconductor industry standard *property specification language* PSL [10].

Our approach to monitoring is invariant under all these semantical choices. As a minimal requirement for being used, the chosen semantics should associate with every formula φ a function $\Omega_\varphi : X^* \to D$ which maps all finite sequences

[6] To be more precise safety properties can be written as positive Boolean combinations of formulae of the form $\Box\varphi$ where φ is a past property, and eventuality properties are negations of safety properties.

into a domain D that contains $\mathbb{B}$ (satisfied/violated) and is augmented with some additional values for undecided formulae.

3.3 Offline and Online Monitoring

In this section we discuss different forms of interaction between the mechanism that generates behaviors and the mechanism that checks whether they satisfy a given property. The behaviors are generated by some kind of a *simulator* that computes states sequentially. Without loss of generality we may assume that the systems we are interested in are *not reverse-deterministic* and, hence, the natural way to generate behaviors is from the past to the future. One may think of three basic modes of interaction (see Fig. 1):

1. *Offline*: The behaviors are completely generated by the simulator before the checking procedure starts. The behaviors are kept in a file which can be read by the monitor in either direction.
2. *Passive Online*: The simulator and the checker run in parallel, with the latter observing the behaviors progressively.
3. *Active Online*: There is a feed-back loop between the generator and the monitor so that the latter may influence the choice of inputs and, hence, the subsequent values of ξ. Such "adaptive" test generation may steer the system toward early detection of satisfaction or violation, and is outside the scope of this paper.

Each behavior is a finite sequence ξ, whose satisfiability value with respect to φ is defined via $\Omega_\varphi(\xi)$ regardless of the checking method. However there are some practical reasons to prefer one method over the other. First, to save time, we would like the checking procedure to reach the most refined conclusions as soon as possible. In the offline setting this will only reduce checking time, while in the online setting the effects of early detection of satisfaction/violation can be much more significant. This is because in certain systems (analog circuits is a notorious example) simulation time is *very long* and if the monitor can abort a simulation once its satisfiability is decided, one can save a lot of time.

The difference between online and offline is, of course, much more significant in situations where monitoring is done with respect to a *physical device*, not its simulated model. We discuss briefly several instances of this situation. The first is when chips are tested after fabrication by injecting real signals to their ports and observing the outcome. Here, the response time of the tester is very important and early (online) detection of violation can have economic importance. In other circumstances we may be monitoring a system which is already up and running. One may think of the supervision of a complex safety-critical plant where the monitoring software should alert the operator about dangerous developments that manifest themselves by property violation or by progress toward such violations. Such a situation calls for online monitoring, although offline monitoring can be used for "post mortem" analysis, for example, analyzing the "black box" after an airplane crash. Monitoring can be used for diagnosis and improvement of non-critical systems as well. For example analyzing whether the behavior of

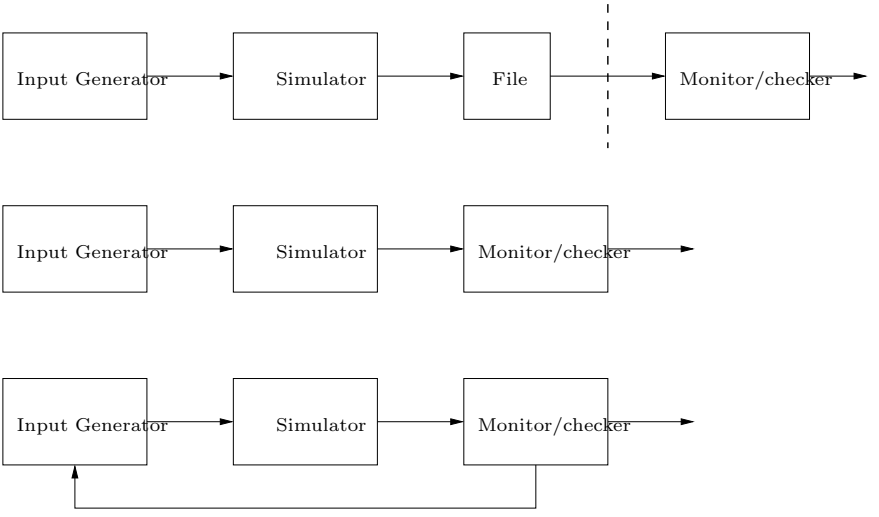


Fig. 1. Offline, passive online and active online modes of interaction between a test generator and a checker

an organization satisfies some specifications concerning the business rules of the enterprise, e.g. "every request if treated within a week". Such an application of monitoring can be done offline by inspecting transaction logs in the enterprise data base.

In the sequel we describe three basic methods for checking satisfaction of LTL formulae by sequences.

The Automaton-Based Method. This is an online-oriented approach that follows the principles used in formal verification. To monitor a property φ we first construct the automaton $\mathcal{A}_\varphi$ that accepts exactly the sequences satisfying φ and then let it read every sequence ξ as it is generated. There is a vast literature concerning the construction of automata from LTL formulae [43] and monitoring does not depend too much on the choice of the translation algorithm. We have, however a preference for the compositional construction, presented in [19] and extended for timed systems in [33]. For each sub-formula ψ of φ, this procedure constructs a sequence $\chi_\psi(\xi)$ indicating the satisfaction of ψ over time, that is $\chi_\psi(\xi)$ has value 1 at t iff $(\xi, t) \models \psi$.

There are two major problems that need to be tackled while employing this method. The first problem is that the natural automaton for φ will be an automaton over *infinite sequences*. This automaton needs to be transformed, via a suitable definition of acceptance conditions, into an automaton over finite sequences that realizes the chosen finitary semantics, as discussed in the previous section. For example, if our satisfiability domain consists of *yes*, *no* and *undecided*, we will output *yes* as soon as the automaton enters a state from which all

the remaining paths are accepting (a positive "sink") and *no* when we enter a negative sink. From all other states the output will be *undecided*.

The second problem is that $\mathcal{A}_\varphi$ is typically non-deterministic. It can be resolved in either of the following ways: 1) Feed the non-deterministic automaton with ξ while keeping track of all the states in which it can be at every time instant. This amounts to performing the classical "subset construction" on-the-fly; 2) Determinize the automaton offline, either using Safra's algorithm for ω-automata [38] or using a simpler algorithm adapted to the finitary semantics.

Purely-Offline Marking. This is the first method we have developed to timed and continuous properties and will be described in more detail in Sect. 6.1. The procedure consists in computing $\chi_\psi(\xi)$ for every sub-formula ψ of φ from the bottom up. It starts with the truth values of propositional formulae $\chi_p(\xi)$ given by the sequence ξ itself. Then, recursively, for each sub-formula ψ with immediate sub-formulae ψ_1 and ψ_2 such that $\chi_{\psi_1}(\xi)$ and $\chi_{\psi_2}(\xi)$ have already been computed, we compute $\chi_\psi(\xi)$ following the semantic rules of LTL. The backward nature of these rules implies that the values of ξ_{ψ_1} and ξ_{ψ_2} at time t will "propagate" to values of ξ_ψ at some $t' \leq t$. The satisfaction function χ_φ for the main formula is computed at the end.

Incremental Marking. This approach combines the simplicity of the offline procedure with the advantages of online monitoring in terms of early detection of violation or satisfaction. After observing a prefix of the sequence $\xi[0, t_1]$ we apply the offline procedure. If, as a result, $\chi_\varphi(\xi)$ is determined at time zero we are done. Otherwise we observe a new segment $\xi[t_1, t_2]$ and then apply the same procedure based on $\xi[0, t_2]$.

A more efficient implementation of this procedure need not start the computation from scratch each time a new segment is observed. It will be often the case that $\chi_\psi(\xi)$ for some sub-formulae ψ is already determined for some subset of $[0, t_1]$ based on $\xi[0, t_1]$. In this case we only need to propagate upwards the new information obtained from $\xi[t_1, t_2]$, combined, possibly, with some additional residual information from the previous segment that was not sufficient for determination in the previous iterations. This procedure will be described in more algorithmic detail in Sect. 6.2.

The choice of the granularity (length of segments) in which this procedure is invoked depends on trade-offs between the computational cost and the importance of early detection.

4 The Timed Level of Abstraction

Coming to export the specification, testing and verification framework from the digital to the analog world, one faces two major conceptual and technical problems [30].

1. The state variables range over subsets of the set of *real numbers* that represent physical magnitudes such as voltage or current;

2. The systems evolve over a *physical* time scale modeled by the real numbers and not over a *logical* time scale defined by a central clock or by events.

Mathematically speaking, the behaviors that should be specified and checked are *signals*, function from $\mathbb{R}_{\geq 0}$ to $\mathbb{R}^n$ rather than *sequences* from $\mathbb{N}$ to $\mathbb{B}^n$ or to some other finite domain. The first problem for monitoring is the problem of how to represent a signal defined over the real time axis inside the computer, given that it is a function defined over an infinite (and non-countable) domain. The very same problem is encountered, of course, by numerical simulators that produce such signals.

Based on our conviction that the dense time problem is more profound than the infinite-state problem we use the following approach. Using a finite number of predicates over the continuous state space, analog signals are transformed into Boolean ones and are checked against properties expressed in a real-time temporal logic whose atomic propositions correspond to those predicates. This allows us to tackle the problem of dense time in isolation. Aspects specific to the continuous state space are discussed in Sect. 7. Note that one can naturally combine these predicates with genuine Boolean propositions to specify properties of hybrid systems (mixed-signal systems in the circuit jargon).

Handling an infinite state space, such as the continuum, using finite formulae is a fundamental mathematical problem. In finite domains one can characterize every individual state by a distinct formula. For example, there is a bijection between $\mathbb{B}^n$ and the set of Boolean terms over $\{p_1, \ldots, p_n\}$ which has one literal for each p_i. The common way to speak of subsets of infinite sets such as $\mathbb{R}^n$ is via *predicates*, functions from $\mathbb{R}^n$ to $\mathbb{B}$, for example inequalities of the form $x_i < d$.

We thus adopt the following approach. Let $\mu_1, \ldots, \mu_m$ be m predicates of the form $\mu : \mathbb{R}^n \to \mathbb{B}$. These predicates define a mapping $M : \mathbb{R}^n \to \mathbb{B}^m$ assigning to every real point a Boolean vector indicating the predicates it satisfies. Applying this mapping in a pointwise fashion to an analog signal $\xi : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ we obtain a Boolean signal $M(\xi) = \xi' : \mathbb{R}_{\geq 0} \to \mathbb{B}^m$ describing the evolution over time of the truth values of these predicates with respect to ξ (see Fig. 2). Events such as *rising* and *falling* in the Boolean signal correspond to some *qualitative* changes in the analog signal, for example threshold crossing of some continuous variable. This is an intermediate level of abstraction where we can observe the *temporal distance* between such events and need to confront the problems introduced by the dense time domain. Timed formalisms such as real-time temporal logics or timed automata are tailored for modeling, specification, verification and monitoring at this level of abstraction, which in addition to its applicability to analog circuits, is also very useful to model phenomena such as delays in digital circuits and execution times of software and, in fact, anything in life that can be modeled as a process where some time has to elapse between its initiation and termination.[7]

[7] It is a pity that the study and utilization of timed models outside academic "formal methods" circles is so negligible compared to their vast, almost universal, domain of application.

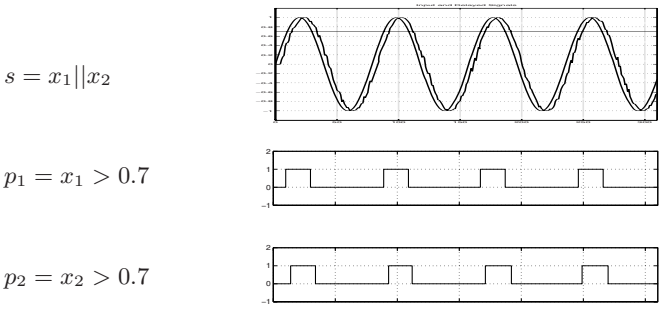


Fig. 2. A 2-dimensional continuous signal and the 2-dimensional Boolean signal obtained from it via the predicates $x_1 > 0.7$ and $x_2 > 0.7$

4.1 Dense-Time Signals: Representation

The major problems in handling Boolean signals by computerized tools are due to the properties of the time domain. In digital systems we have the *discrete order* $(\mathbb{N}, <)$, which means that there is a relation (successor) that generates the whole order relation. In other words, for every t and t' such that $t < t'$, there is a finite positive k such that $t' = Suc^k(t)$. This also implies that whenever we put a bound r on the range of the time variable, the set $\{t : 0 \leq t \leq r\}$ is *finite* and every behavior defined on the interval $[0, r]$ can be represented by a finite set $\{\xi[0], \xi[1], \ldots, \xi[r]\}$.

The dense order $(\mathbb{R}, <)$ does not admit such a property, and for every $t < t'$ one can find t'' such that $t < t'' < t'$. This implies that in order to specify a dense-time signal, even if restricted to a bounded time interval $[0, r]$, one might need to specify an *infinite* set of values. For arbitrary analog signals the only way to provide these values throughout the entire interval is via analytic expressions such as $\xi[t] = \sin(t)$. Otherwise an analog signal can only be partially represented by its values at a finite subset of the time domain consisting of *sampling points* (more on that in Sect. 7). As for Boolean signals, let us note that functions from $\mathbb{R}_+$ to $\mathbb{B}$ can be rather weird objects, potentially switching between 0 and 1 infinitely many times in a bounded interval of time (the so-called Zeno phenomenon).[8] From now on we restrict our attention to non-Zeno Boolean signals.

A non-Zeno Boolean signal ξ defined over an interval $[0, r)$ decomposes naturally into a finite sequence of intervals $I_0, I_1, \ldots, I_k$ such that $I_0 = [0, t_1)$, $I_i = [t_i, t_{i+1})$, $I_k = [t_{k-1}, r)$, the value of ξ is constant in every interval, and $\xi(I_{i+1}) = \neg\xi(I_i)$. The set of intervals, together with the value at $t = 0$ determine the value of ξ at *any* point and can serve as a basis for checking properties relative to ξ.

[8] Such Zeno signals can be obtained from analog signals via Booleanization: just consider a signal representing a damped oscillation around zero and its Boolean image via the predicate $x < 0$.

4.2 Dense-Time Signals: Properties

The temporal operators of LTL are of two types. The *next* operator is bounded and quantitative. It specifies something that should happen within the very next step or, if used iteratively, within a bounded number of steps. The *until* operator and its derivatives are unbounded and qualitative, requiring that something should or should not hold at some unspecified future instant. The latter properties are not affected seriously from the passage to dense time, while quantitative operators need to be redefined. To start with, the *next* operator which specifies at t what should hold at the *smallest* t' such the $t < t'$ becomes meaningless. Instead one has to use operators that specify at t what should hold at time $t + d$ or during the interval $t \oplus [a, b] = [t + a, t + b]$. Many temporal logics over such metric time have been proposed and studied [21,4,17,18] and we will focus on the logic MITL, which is a natural adaptation of LTL to dense time [3].

Dense time also has an influence on the different monitoring procedures. As we shall see, the offline procedure based on marking the truth values of subformulae over time, can be rather easily adapted to signals. However the online approaches are more problematic. Consider the approach based on translating a formula into an automaton that accepts its models. The appropriate automaton will be a timed automaton, which reads signals continuously and uses auxiliary clock variables to measure times since the occurrence of certain events. Automata corresponding to MITL formulae are, more often than not, non-deterministic, a feature that, in a discrete-time framework, can be resolved using subset construction, either offline or on the fly. Dense non-determinism is another story as the automaton may stay during an interval in a state q while at any moment during the interval it may take a transition to q', thus spawning uncountably-many runs of the automaton. The impossibility of an offline determinization of timed automata is a well-known fact in the domain, but in Sect. 6.3 we will mention some remedies to this problem.

We can now move to more detailed definitions of signals and their corresponding temporal logics, followed by the description of their monitoring algorithms.

5 Boolean Signals and Their Temporal Logics

5.1 Signals

Two basic semantic domains can be used to describe timed behaviors. *Time-event sequences* consist of instantaneous events separated by time durations while discrete-valued *signals* are functions from time to some discrete domain. The reader may consult the introduction to [6] for more details on the algebraic characterization of these domains. In this work we use Boolean signals as the semantic domain, which is the natural choice, both for the logic MITL and the circuit application domain.

Let the time domain T be the set $\mathbb{R}_{\geq 0}$ of non-negative real numbers. A Boolean signal is a function $\xi : T \to \mathbb{B}^n$. We use $\xi[t]$ for the value of the signal at time t and the notation $\sigma_1^{t_1} \cdot \sigma_2^{t_2} \cdots$ for a signal whose value is σ_1 at the interval $[0, t_1)$,

σ_2 in the interval $[t_1, t_1 + t_2)$, etc. A signal whose value is defined only on an interval $[0, r)$ is called finite and of *metric*[9] length r (denoted by $|\xi| = r$). The restriction of a signal to length d is defined as

$$\xi' = \langle \xi \rangle_d \text{ iff } \xi'[t] = \begin{cases} \xi[t] \text{ if } t < d \\ \perp \quad \text{otherwise} \end{cases}$$

For the sake of simplicity we restrict ourselves to *left-closed right-open* signal segments and to timed modalities that use only closed intervals. As a consequence we exclude signals with *punctual* "intervals" which are meaningless in the algebraic definition of signals [6,5]. The more general case was treated in [3].

Different Boolean signals can be combined and separated using the standard operations of *pairing* and *projection* defined as

$$\xi_1 \mid\mid \xi_2 = \xi_{12} \text{ if } \forall t \ \xi_{12}[t] = (\xi_1[t], \xi_2[t])$$
$$\xi_1 = \pi_1(\xi_{12}) \quad \xi_2 = \pi_2(\xi_{12})$$

In particular, $\pi_p(\xi)$ will denote the projection of ξ on the dimension that corresponds to proposition p.

Any Boolean operation OP can be "lifted" to an operation on signals as

$$\xi = \text{OP}(\xi_1, \xi_2) \text{ iff } \forall t \ \xi[t] = \text{OP}(\xi_1[t], \xi_2[t])$$

When we apply operations on signals of different lengths we use the convention

$$\text{OP}(v, \perp) = \text{OP}(\perp, v) = \perp$$

which guarantees that if $\xi = \text{OP}(\xi_1, \xi_2)$ then $|\xi| = \min(|\xi_1|, |\xi_2|)$.

Any reasonable Boolean signal can be represented using a countable number of intervals. An *interval covering* of a given interval $I = [0, r)$ is a sequence $\mathcal{I} = I_1, I_2 \ldots$ of left-closed right-open intervals such that $\bigcup I_i = I$ and $I_i \cap I_j = \emptyset$ for every $i \neq j$. An interval covering $\mathcal{I}'$ is said to *refine* $\mathcal{I}$, denoted by $\mathcal{I}' \prec \mathcal{I}$ if $\forall I' \in \mathcal{I}' \ \exists I \in \mathcal{I}$ such that $I' \subseteq I$.

An interval covering $\mathcal{I}$ is said to be *consistent* with a signal ξ if $\xi[t] = \xi[t']$ for every t, t' belonging to the same interval I_i. In that case we can use the notation $\xi(I_i)$. Clearly, if $\mathcal{I}$ is consistent with ξ, so is any $\mathcal{I}' \prec \mathcal{I}$. We restrict ourselves to signals of *finite variability*, that is, signals admitting a finite consistent interval covering. We denote by $\mathcal{I}_\xi$ the *minimal* interval covering consistent with a finite variability signal ξ. The set of positive intervals of ξ is $\mathcal{I}_\xi^+ = \{I \in \mathcal{I}_\xi : \ \xi(I) = 1\}$ and the set of negative intervals is $\mathcal{I}_\xi^- = \mathcal{I}_\xi - \mathcal{I}_\xi^+$.

A signal ξ is said to be *unitary* if $\mathcal{I}_\xi^+$ is a singleton. Any finite-variability signal ξ over a bounded interval can be decomposed into a union of k unitary signals such that $\xi = \xi^1 \vee \ldots \vee \xi^k$, see Fig. 3.

The *concatenation* $\xi = \xi_1 \cdot \xi_2$ of two signals ξ_1 and ξ_2 defined over the intervals $[0, r_1)$ and $[0, r_2)$ respectively is a signal over $[0, r_1 + r_2)$ defined as:

$$\xi[t] = \begin{cases} \xi_1[t] & \text{if } t < r_1 \\ \xi_2[t - r_1] & \text{otherwise} \end{cases}$$

[9] To distinguish it from the *logical* length which corresponds to the number of state changes.

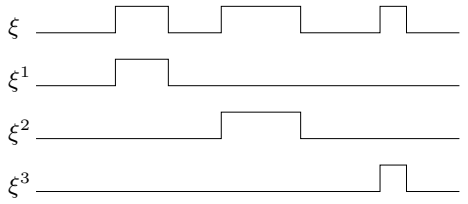


Fig. 3. A signal ξ and its unitary decomposition (ξ^1, ξ^2, ξ^3)

The *d-suffix* of a signal ξ is the signal $\xi' = d\backslash\xi$ obtained from ξ by removing the prefix $\langle\xi\rangle_d$ from ξ, that is,

$$\xi'[t] = \xi[t+d] \text{ for every } t \in [0, |\xi| - d).$$

The *Minkowski sum* and *difference* of two sets P_1 and P_2 are defined as

$$P_1 \oplus P_2 = \{x_1 + x_2 : x_1 \in P_1, x_2 \in P_2\}$$
$$P_1 \ominus P_2 = \{x_1 - x_2 : x_1 \in P_1, x_2 \in P_2\}.$$

Of particular interest are the applications of these operations to one-dimensional sets consisting of elements of the time domain T:

$$\{t\} \oplus [a,b] = [t+a, t+b], \qquad [m,n) \oplus [a,b] = [m+a, n+b)$$

$$\{t\} \ominus [a,b] = [t-b, t-a], \qquad [m,n) \ominus [a,b] - [m-b, n-a)$$

The operation that will be used for computing the satisfiability of a formula whose major operator is a bounded temporal operator is the operation of *back shifting*.

Definition 1 (Back Shifting). *The $[a,b]$-back-shifting of a Boolean signal ξ', denoted by $\xi = \textsc{Shift}_{[a,b]}(\xi')$, is a signal ξ such thet for every t, $\xi[t] = 1$ iff there exists $t' \in t \oplus [a,b]$ such that $\xi'[t'] = 1$.*

The resemblance of this definition to the semantics of the $\Diamond_{[a,b]}$ operator (to be defined in Sect. 5.2) is not a coincidence. If $\varphi = \Diamond_{[a,b]}\varphi'$ then the respective satisfiability signals of φ and φ' satisfy $\chi_\varphi = \textsc{Shift}_{[a,b]}(\chi_{\varphi'})$. This operation is easy to compute on a representation based on an interval covering of the signals. When ξ' is a unitary signal with $\mathcal{I}^+_{\xi'} = \{I'\}$, the result of back shifting is the unitary signal ξ with $\mathcal{I}^+_\xi = \{I\}$ where $I = I' \ominus [a,b] \cap T$ (the intersection with T is needed to remove negative values, see Fig. 4).

5.2 Real-Time Temporal Logic

The syntax of MITL is defined by the grammar

$$\varphi := p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \mathcal{U}_{[a,b]} \varphi_2 \mid \varphi_1 \mathcal{U} \varphi_2$$

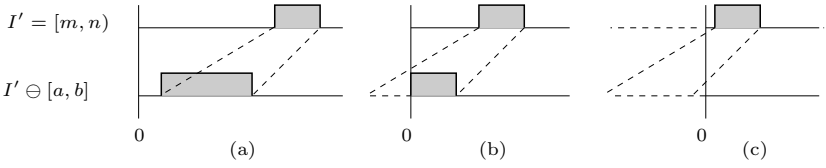


Fig. 4. Three instances of back shifting $I = [m, n) \ominus [a, b]$: (a) $I = [m - b, n - a)$; (b) $I = [0, n - a]$ because $m - b < 0$; (c) $I = \emptyset$ because $n - a < 0$

where p belongs to a set $P = \{p_1, \ldots, p_n\}$ of propositions and $b > a \geq 0$ are rational numbers.[10] From basic MITL operators one can derive other standard Boolean and temporal operators, in particular the time-constrained *eventually* and *always* operators:

$$\Diamond_{[a,b]}\varphi = \mathrm{T}\ \mathcal{U}_{[a,b]}\varphi \quad \text{and} \quad \Box_{[a,b]}\varphi = \neg\Diamond_{[a,b]}\neg\varphi$$

We interpret MITL over n-dimensional Boolean signals and define the satisfiability relation similarly to LTL.

$$\begin{array}{ll}
(\xi, t) \models p & \leftrightarrow p[t] = \mathrm{T} \\
(\xi, t) \models \neg\varphi & \leftrightarrow (\xi, t) \not\models \varphi \\
(\xi, t) \models \varphi_1 \vee \varphi_2 & \leftrightarrow (\xi, t) \models \varphi_1 \text{ or } (\xi, t) \models \varphi_2 \\
(\xi, t) \models \varphi_1 \mathcal{U} \varphi_2 & \leftrightarrow \exists t' \geq t\ (\xi, t') \models \varphi_2 \text{ and} \\
 & \quad\ \forall t'' \in [t, t'], (\xi, t'') \models \varphi_1 \\
\xi, t) \models \varphi_1 \mathcal{U}_{[a,b]} \varphi_2 & \leftrightarrow \exists t' \in t \oplus [a, b]\ (\xi, t') \models \varphi_2 \text{ and} \\
 & \quad\ \forall t'' \in [t, t'], (\xi, t'') \models \varphi_1
\end{array}$$

$$\begin{array}{ll}
(\xi, t) \models \Diamond_{[a,b]}\varphi & \leftrightarrow \exists t' \in t \oplus [a, b]\ (\xi, t') \models \varphi \\
(\xi, t) \models \Box_{[a,b]}\varphi & \leftrightarrow \forall t' \in t \oplus [a, b]\ (\xi, t') \models \varphi
\end{array}$$

The past version of MITL is obtained by replacing the $\mathcal{U}_{[a,b]}$ operator by the *since* operator $\mathcal{S}_{[a,b]}$, from which one can derive the time-constrained *sometime in the past* ($\Diamond\!\!\!-$) and *always in the past* ($\Box\!\!\!-$), operators. The semantics of the past operators is defined as

$$\begin{array}{ll}
(\xi, t) \models \varphi_1 \mathcal{S}_{[a,b]} \varphi_2 & \leftrightarrow \exists t' \in t \ominus [a, b]\ (\xi, t') \models \varphi_2 \text{ and} \\
 & \quad\ \forall t'' \in [t', t], (\xi, t'') \models \varphi_1
\end{array}$$

$$\begin{array}{ll}
(\xi, t) \models \Diamond\!\!\!-_{[a,b]}\varphi & \leftrightarrow \exists t' \in t \ominus [a, b]\ (\xi, t') \models \varphi \\
(\xi, t) \models \Box\!\!\!-_{[a,b]}\varphi & \leftrightarrow \forall t' \in t \ominus [a, b]\ (\xi, t') \models \varphi
\end{array}$$

In this paper we focus on the more difficult future fragment of MITL.

6 Checking Timed Properties

In this section we describe two procedures for checking MITL properties:

[10] In fact, it is sufficient to consider integer constants.

1. An offline marking procedure that propagates truth values upwards from propositions via super-formulae up to the main formula. This procedure has been first presented in [31].
2. An incremental marking procedure that updates the marking each time a new segment of the signal is observed. This procedure is described in [37].

A central notion in all these algorithms is that of the *satisfaction signal* $\xi' = \chi_\varphi(\xi)$ associated with a formula φ and a signal ξ. In this signal $\xi'[t] = 1$ whenever $(\xi, t) \models \varphi$. We remind the reader that due to non-causality the value of $\xi'[t]$ is not necessarily known at time t, that is, after observing $\xi[t]$, and may depend on future values of ξ. Whenever the identity of ξ is clear from the context, we will use the shorthand notation χ_φ.

6.1 Offline Marking

This algorithm [31] works as follows. It has as input a formula φ and an n-dimensional Boolean signal of length r. For every sub-formula ψ of φ it computes its satisfiability signal $\chi_\psi(\xi)$. To simplify the discussion we restrict the presentation to a bounded version of MITL where the unbounded *until* is not used. Hence we have properties that are fully determined if the signal is sufficiently long. In the case where the signal is too short the output is *undecided*, denoted by $\perp$. The procedure is recursive on the structure (parse tree) of the formula. It goes down until the propositional variables whose values are determined directly by ξ, and then propagates values as it comes up from the recursion. We will use OP_1 and OP_2 for arbitrary unary and binary logical or temporal operators. As a preparation for the incremental version, we do not pass ξ and χ_φ as input or output parameters but rather store them in global data structures.

Algorithm 1. OFFLINEMITL

```
input : an MITL Formula φ
switch φ do
    case p
        χ_φ := π_p(ξ);
    end
    case OP1(φ1)
        OFFLINEMITL (φ1);
        χ_φ := COMBINE(OP1, φ1);
    end
    case OP2(φ1, φ2)
        OFFLINEMITL (φ1);
        OFFLINEMITL (φ2);
        χ_φ := COMBINE (OP2, χ_φ1, χ_φ2);
    end
end
```

Most of the work in this algorithm is done by the COMBINE function which for $\varphi = \text{OP}_2(\varphi_1, \varphi_2)$ computes χ_φ from the signals χ_{φ_1} and χ_{φ_2}, which may differ

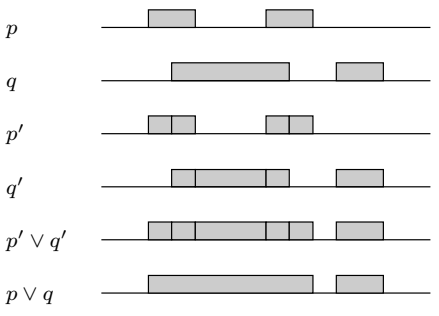


Fig. 5. To compute $p \vee q$ we first refine the interval covering to obtain the semantically-equivalent representations p' and q'. We then perform interval-wise operations to obtain $p' \vee q'$ and then merge adjacent positive intervals

in length. We describe briefly how this function works for each of the operators, with a sufficient detail to understand how it operates on the representation of the input and output signals by their sets of positive intervals. For the sake of readability we omit the description of various mundane optimizations.

$\chi_\varphi := \mathbf{Combine}(\neg, \chi_{\varphi_1})$ The negation is computed by simply changing the Boolean value of each minimal interval in the representation of χ_{φ_1}.

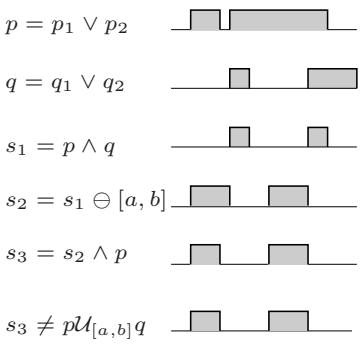
$\chi_\varphi := \mathbf{Combine}(\vee, \chi_{\varphi_1}, \chi_{\varphi_2})$ For the disjunction we first construct a refined interval covering $\mathcal{I} = \{I_1, \ldots, I_k\}$ for $\chi_{\varphi_1} || \chi_{\varphi_2}$ so that the mutual values of both signals become uniform in every interval. Then we compute the disjunction interval-wise, that is, $\varphi(I_i) = \varphi_1(I_i) \vee \varphi_2(I_i)$. Finally we merge adjacent intervals having the same Boolean value to obtain the minimal interval covering $\mathcal{I}_{\chi_\varphi}$. This procedure is illustrated in Fig. 5.

$\chi_\varphi := \mathbf{Combine}(\Diamond_{[a,b]}, \chi_{\varphi_1})$ This is the most important part of our procedure which computes $\chi_\varphi := \textsc{Shift}_{[a,b]}(\xi_{\varphi_1})$. For every positive interval $I \in {\mathcal{I}^+}_{\varphi_1}$ we compute its back shifting $I \ominus [a,b] \cap T$ and insert it to ${\mathcal{I}^+}_{\varphi}$. Overlapping positive intervals in ${\mathcal{I}^+}_{\varphi}$ are merged to obtain a minimal consistent interval covering. In the process, all the negative intervals shorter than $b-a$ disappear.[11]

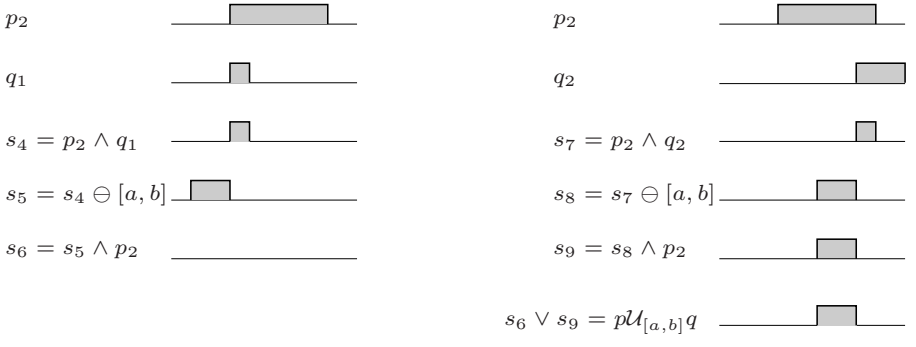
$\chi_\varphi := \mathbf{Combine}(\mathcal{U}_{[a,b]}, \chi_{\varphi_1}, \chi_{\varphi_2})$ The implementation of the timed *until* operator is based on the equivalence $\varphi_1 \mathcal{U}_{[a,b]} \varphi_2 \leftrightarrow (\Diamond_{[a,b]}(\varphi_1 \wedge \varphi_2)) \wedge \varphi_1$ when χ_{φ_1} is a unitary signal. This is because for a unitary signal, if φ_1 holds at t_1 and at t_2 it must hold during the whole interval. This does not hold for arbitrary signals, see Fig. 6. In order to treat the general case where χ_{φ_1} is a non-unitary signal we first need to decompose it into the unitary signals $\chi^1_{\varphi_1}, \ldots, \chi^k_{\varphi_1}$ and then compute

$$\chi^i_\varphi = (\textsc{Shift}_{[a,b]}(\chi^i_{\varphi_1} \wedge \chi_{\varphi_2})) \wedge \chi^i_{\varphi_1}$$

[11] This procedure can be viewed alternatively as shifting the *negative* intervals by $[b,a]$.



(a)



(b)

Fig. 6. Computing satisfiability of $p\mathcal{U}_{[a,b]}q$ via the satisfiability of $\Diamond_{[a,b]}(q \wedge p) \wedge p$. (a) wrong results obtained with non-unitary signals; (b) correct results obtained with a unitary decomposition $p = p_1 \vee p_2$ and $q = q_1 \vee q_2$. The computation with p_1 is omitted as it has an empty intersection with q

for each $i \in [1, k]$. Finally we recompose the resulting signals as

$$\chi_\varphi = \bigvee_{i=1}^{k} \chi_\varphi^i.$$

6.2 Incremental Marking

Incremental marking is performed using a kind of piecewise-online procedure invoked each time a new segment of ξ, denoted by Δ_ξ, is observed. For each

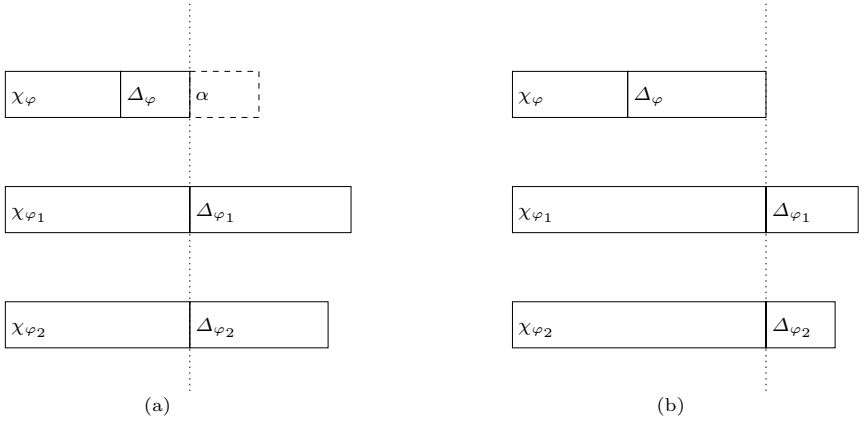


Fig. 7. A step in an incremental update: (a) A new segment α for φ is computed from Δ_{φ_1} and Δ_{φ_2}; (b) α is appended to Δ_φ and the endpoints of χ_{φ_1} and χ_{φ_1} are shifted forward accordingly

sub-formula ψ the algorithm stores its already-computed satisfaction signal partitioned into a concatenation of two signals $\chi_\psi \cdot \Delta_\psi$ with χ_ψ consisting of values already propagated to the super-formula of ψ, and Δ_ψ consists of values that have already been computed but which have not yet been propagated to the super-formula and can still influence its satisfaction.

Initially all signals are empty. Each time a new segment Δ_ξ is read, a recursive procedure similar to the offline procedure is invoked, which updates every χ_ψ and Δ_ψ from the bottom up. The difference with respect to the offline algorithm is that only the segments of the signal that have not been propagated upwards participate in the update of their super-formulae. This may result in a lot of saving when the signal is very long, as has been demonstrated empirically in [37].

As an illustration consider $\varphi = \text{OP}(\varphi_1, \varphi_2)$ and the corresponding truth signals of Fig. 7-(a). Before the update we always have $|\chi_\varphi \cdot \Delta_\varphi| = |\chi_{\varphi_1}| = |\chi_{\varphi_2}|$: the parts Δ_{φ_1} and Δ_{φ_2} that may still affect φ are those that start at the point from which the satisfaction of φ is still unknown. We apply the COMBINE procedure on Δ_{φ_1} and Δ_{φ_2} to obtain a new (possibly empty) segment α of Δ_φ. This segment is appended to Δ_φ in order to be propagated upwards, but before that we need to shift the borderline between χ_{φ_1} and Δ_{φ_1} (as well as between χ_{φ_2} and Δ_{φ_2}) in order to reflect the update of Δ_φ. The procedure is described in Algorithm 2.

6.3 Monitoring Using Timed Automata

Our contribution to the automaton-based approach for checking timed properties will be described elsewhere and we mention the relevant results briefly. In [32] we have shown how to build deterministic timed automata from past MITL properties and gave an alternative proof of the impossibility to do so for future MITL. The difference in dererminizability between the past and future

Algorithm 2. INC-OFFLINE-MITL

input : an MITL Formula φ and an increment Δ_ξ of a signal

switch φ **do**
  case p
    $\Delta_\varphi := \Delta_\varphi \cdot \pi_p(\Delta_\xi)$;
  end
  case $\text{OP}_1(\varphi_1)$
    INC-OFFLINE-MITL (φ_1, Δ_ξ);
    $\alpha :=$ COMBINE$(\text{OP}_1, \Delta_{\varphi_1})$;
    $d := |\alpha|$;
    $\Delta_\varphi := \Delta_\varphi \cdot \alpha$;
    $\chi_{\varphi_1} := \chi_{\varphi_1} \cdot \langle \Delta_{\varphi_1} \rangle_d$;
    $\Delta_{\varphi_1} := d \backslash \Delta_{\varphi_1}$
  end
  case $\text{OP}_2(\varphi_1, \varphi_2)$
    INC-OFFLINE-MITL (φ_1, Δ_ξ);
    INC-OFFLINE-MITL (φ_2, Δ_ξ);
    $\alpha :=$ COMBINE$(\text{OP}_2, \Delta_{\varphi_1}, \Delta_{\varphi_2})$;
    $d := |\alpha|$;
    $\Delta_\varphi := \Delta_\varphi \cdot \alpha$;
    $\chi_{\varphi_1} := \chi_{\varphi_1} \cdot \langle \Delta_{\varphi_1} \rangle_d$;
    $\Delta_{\varphi_1} := d \backslash \Delta_{\varphi_1}$;
    $\chi_{\varphi_2} := \chi_{\varphi_2} \cdot \langle \Delta_{\varphi_2} \rangle_d$;
    $\Delta_{\varphi_2} := d \backslash \Delta_{\varphi_2}$
  end
end

fragments turned out to be a syntactical accident *not related* to the difference in the causality between past and future (note that the logic MTL, admitting *punctual* modalities such as $\Diamond_d$ is non-deterministic in both directions). The reason is that interval-based modalities, when they point backwards, erase the effect of small fluctuations in Boolean signals, see [32]. In [33] we adapted the compositional construction of non-deterministic automata from LTL [19] to MITL. Finally we have shown in [34] how to construct deterministic timed automata for the *bounded* fragment of the more general logic MTL under *bounded variability* assumptions. More technical details concerning the techniques used can be found in those papers. We mention the works of [42,24] and [12,13,14] which inspired part of our work.

7 Continuous Signals

The algorithms developed for dense-time Boolean signals, provide a solid basis for monitoring continuous signals when the properties belong to the *signal temporal logic* (STL) [31,37] which is nothing but MITL, parameterized by a set of numerical predicates playing the role of atomic propositions. For such properties, each continuous signal is transformed, via the numerical predicates appearing in

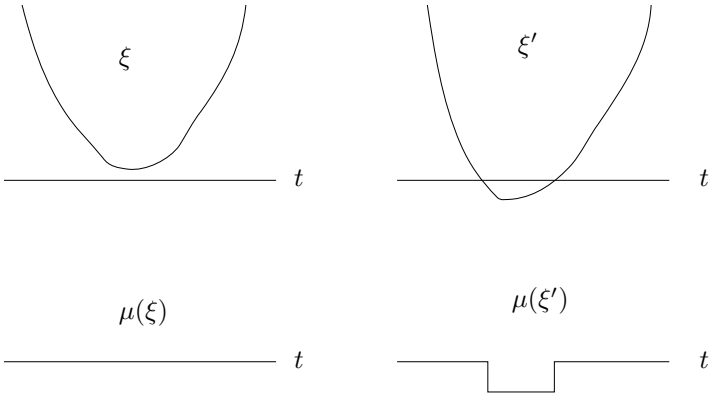


Fig. 8. Two signals which are close from a continuous point of view, one satisfying the property $\Box(x > 0)$ and one violating it

the property, into a Boolean signal which is checked against the MITL "skeleton" of the formula. In the rest of this section we discuss technical problems related to the applicability of the "Booleanization" procedure.

As we have seen, non-Zeno Boolean signals, albeit the fact that they are defined over dense time domain, admit an *exact finite representation* via the switching points that define their *true* and *false* intervals. This is no longer the case for continuous signals where we have a contrast between the *ideal mathematical object*, consisting of an uncountable number of pairs $(t, \xi[t])$ with t ranging over some interval $[0, r) \subseteq \mathbb{R}_{\geq 0}$, and any *finite* representation which consists of a collection of such pairs, with t restricted to range over a finite set of *sampling points*. The values of ξ at sampling points t_1 and t_2 may, at most, impose some constraints on the values of ξ inside the interval (t_1, t_2). Such constraints can be based on the dynamics of the generating system and the manner in which the numerical simulator produces the signal values at the sampling points. Numerical analysis is a very mature domain with a lot of accumulated experience concerning tradeoffs between accuracy and computation time. Its major premise is that given a model of the system as a continuous dynamical system defined by a differential equation[12], one can improve the quality of a discrete-time approximation of its behavior by employing denser sets of sampling points and more sophisticated numerical integration procedures.[13]

In order to speak quantitatively about the approximation of a signal by another we need the concept of a *distance/metric* imposed on the space of continuous signals. A metric is a function that assigns to two signals ξ_1 and ξ_2 a non-negative value $\rho(\xi_1, \xi_2)$ which indicates how they resemble each other. Using metrics one can express the "convergence" of a numerical integration scheme as

[12] It is worth noting that some models used for rapid simulation of transistor networks cannot always be viewed as continuous dynamical systems in the classical mathematical sense.

[13] For systems which are stable the quality can be improved indefinitely.

the condition that $\lim_{d\to 0} \rho(\xi, \xi_d) = 0$ where ξ is the ideal mathematical signal and ξ_d is its numerical approximation using an integration step d.

Metrics and norms for continuous signals are used extensively in circuit design, control and signal processing. There are, however, major problems concerned with their application to property monitoring due to the incompatibility between the continuous nature of the signals and the discrete nature of $\{0, 1\}$-properties, a phenomenon which is best illustrated using the following simple example. Consider the property $\Box(x > 0)$ and an ideal mathematical signal ξ that satisfies the property but which passes very close to zero at some points. We can easily transform ξ into a signal ξ' which is *very close* to ξ under any reasonable continuous metric, but according to the metric induced by the property, these signals are as distant as can be: one of them satisfies the property and the other violates it (see Fig. 8).

Moreover, if the sojourn time of a signal below zero is short, an arbitrary shift in the sampling can make the monitor miss the zero-crossing event and declare the signal as satisfying (see Fig. 9). In this sense properties are not *robust* as small variations in the signal may lead to large variations in its property satisfaction. Let us mention some interesting ideas due to P. Caspi [20] concerning new metrics for bridging the gap between the continuous and the discrete points of view. Such metrics are expressible, by the way, in STL [37].

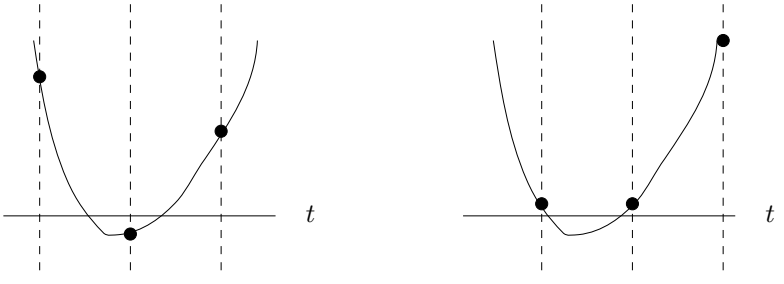


Fig. 9. Shifting the sampling points, zero crossing can be missed

The abovementioned issues can be handled pragmatically in our context, without waiting for a completely-satisfactory theoretical solution to this fundamental problem. The following assumptions facilitate the monitoring of sampled continuous signals against STL properties, passing through the timed abstraction:

1. *Sufficiently-dense sampling*: the simulator detects every change in the truth value of any of the predicates appearing in the formula at a sufficient accuracy. This way the positive intervals of all the Boolean signals that correspond to these predicates are determined. This requirement imposes some level of sophistication on the simulator that has to perform several back-and-forth iterations to locate the time instances where a threshold crossing occurs. Many simulation tools used in industry have already such event-detection features. A survey of the treatment of discontinuous phenomena by numerical simulators can be found in [26].

2. *Bounded variability*: some restrictive assumptions can be made about the values of the signal between two sampling points t_1 and t_2. For example one may assume that ξ is monotone so that if $\xi[t_1] \leq \xi[t_2]$ then $\xi[t'_1] \leq \xi[t'_2]$ for every t'_1 and t'_2 such that $t_1 < t'_1 < t'_2 < t_2$. An alternative condition could be a condition a-la Lipschitz: $|\xi[t_2] - \xi[t_1]| \leq K|t_2 - t_1|$. Such conditions guarantee that the signal does not get wild between the sampling points, otherwise property checking based on these values is useless.

Under such assumptions every continuous signal which is given by a discrete-time representation, based on sufficiently-dense sampling, induces a well-defined Boolean signal ready for MITL monitoring. Let us add at this point a general remark that the standards of exactness and exhaustiveness as maintained in discrete verification cannot and should not be exported to the continuous domain, and even if we are not guaranteed that all events are detected, we can compensate for that by using safety margins in the predicates and properties.

8 Monitoring STL Properties

In this section we illustrate the monitoring of STL properties against signals produced by the numerical simulator Matlab/Simulink, used mainly for control and signal-processing applications, but also for modeling analog circuits at the functional level of abstraction. The waveforms presented here are the output of our first prototype of analog monitoring tool, which parses STL properties and applies the offline marking procedure described in Sect. 6.1.

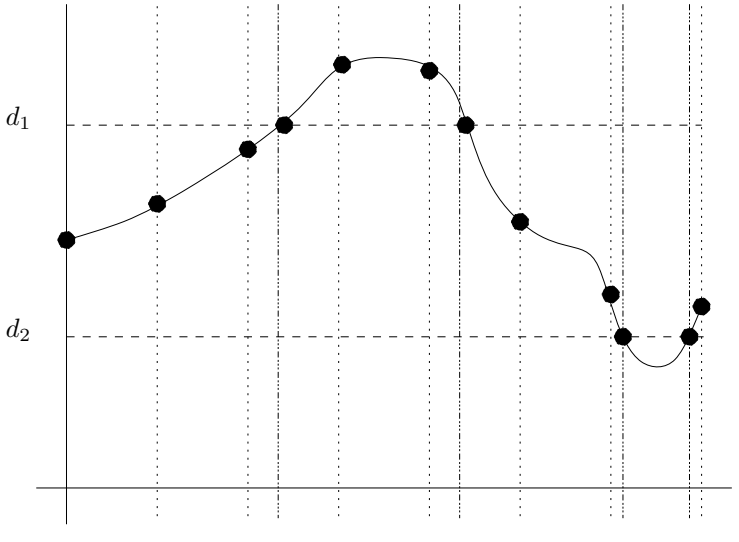


Fig. 10. Sufficiently-dense sampling with respect to the two thresholds d_1 and d_2. The set of sampling points consists of a uniform grid augmented with the threshold-crossing points

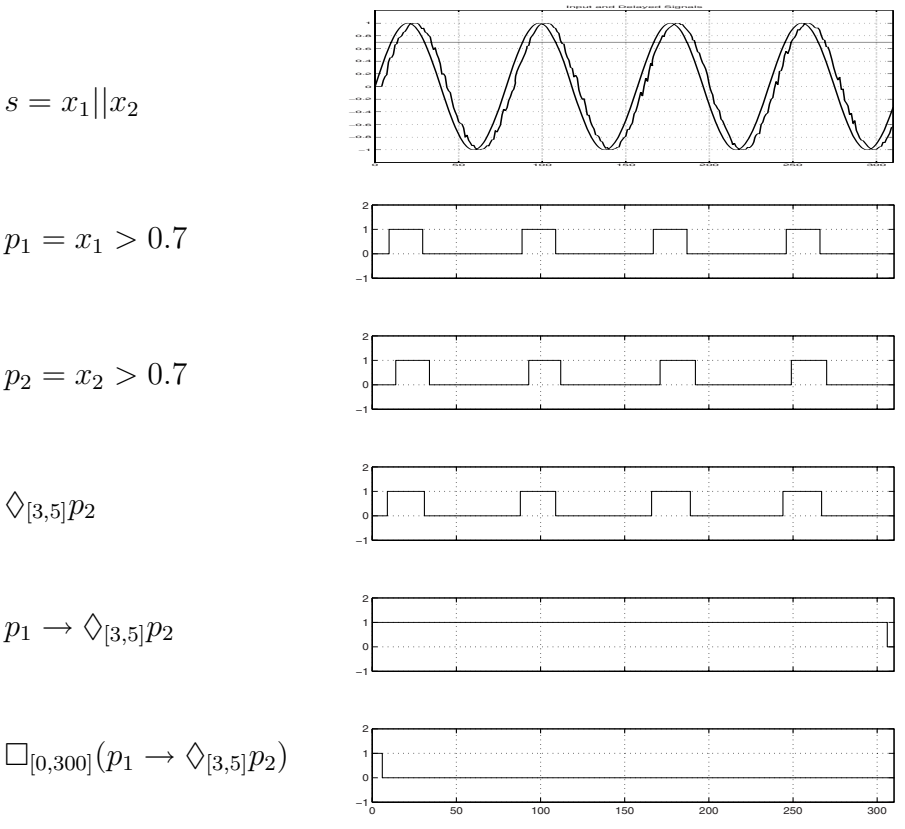


Fig. 11. A 2-dimensional signal satisfying the property $\Box_{[0,300]}((x_1 > 0.7) \Rightarrow \Diamond_{[3,5]}(x_2 > 0.7))$. Boolean signals correspond to the evolution of the truth values of sub-formulae over time

8.1 Following a Reference Signal

As a first example consider the property

$$\varphi_1 : \quad \Box_{[0,300]}((x_1 > 0.7) \Rightarrow \Diamond_{[3,5]}(x_2 > 0.7))$$

which requires that whenever x_1 crosses the threshold 0.7, so does x_2 within $t \in [3, 5]$ time units. We fix x_1 to be the sinusoid

$$x_1[t] = \sin(\omega t),$$

and let x_2 be a signal generated by

$$x_2[t] = \sin(\omega(t + d)) + \theta$$

where d is a random delay ranging in $[3, 5]$ degrees and θ is an additive random noise. The marking procedure is illustrated in Fig. 11. The Boolean signals corresponding to the atomic propositions p_1 and p_2 are derived from the sampled

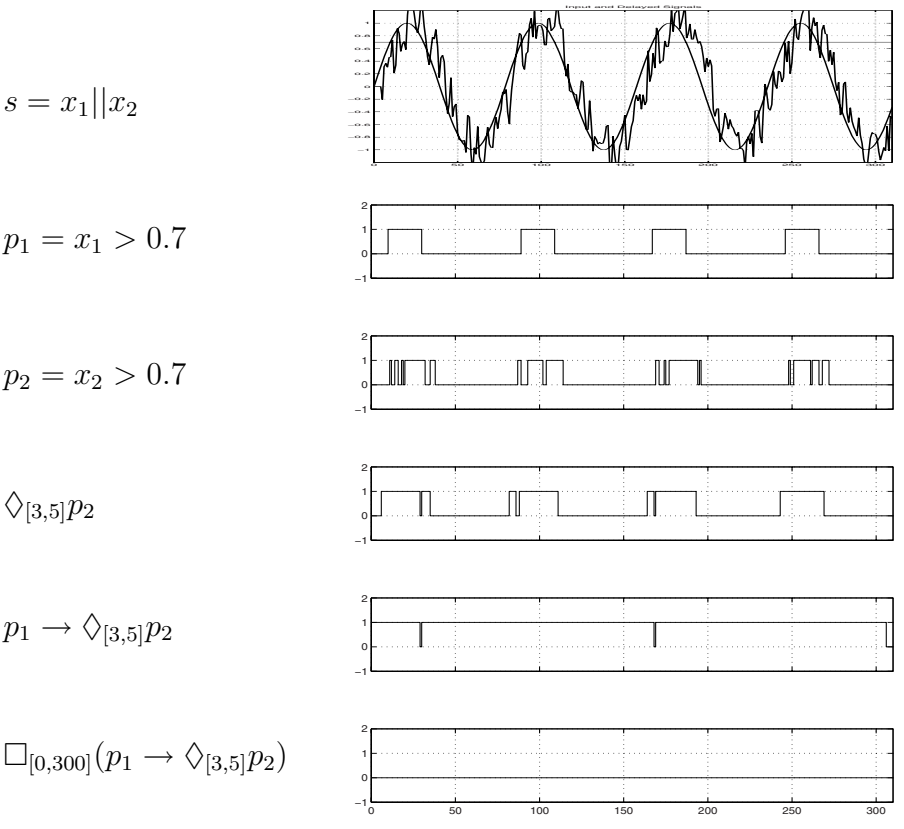


Fig. 12. A 2-dimensional signal violating the property $\Box_{[0,300]}((x_1 > 0.7) \Rightarrow \Diamond_{[3,5]}(x_2 > 0.7))$

analog signal. From there the truth values of the sub-formulae $\Diamond_{[3,5]}(x_2 > 0.7)$, $(x_1 > 0.7) \Rightarrow \Diamond_{[3,5]}(x_2 > 0.7)$ are marked as intermediate steps toward the marking of φ_1 which is satisfied in this example. In Fig. 12 we apply the same procedure to check φ_1 against a signal in which x_2 was generated with a much larger additive noise $\theta \in [-0.5, 0.5]$. The fluctuations in the value of x_2 are reflected in the Boolean abstraction p_2 and lead to a violation of the property at some points where $x_1 > 0.7$ is not followed by $x_2 > 0.7$ within the pre-specified delay.

8.2 Stabilizability

The second example is a very typical stabilizability property used extensively in control and signal processing. The system in question is supposed to maintain a controlled variable y around a fixed level despite disturbances x coming from the outside world. The actual system used to generate this example is a water-level controller for a nuclear plant. The disturbances come from changes in the

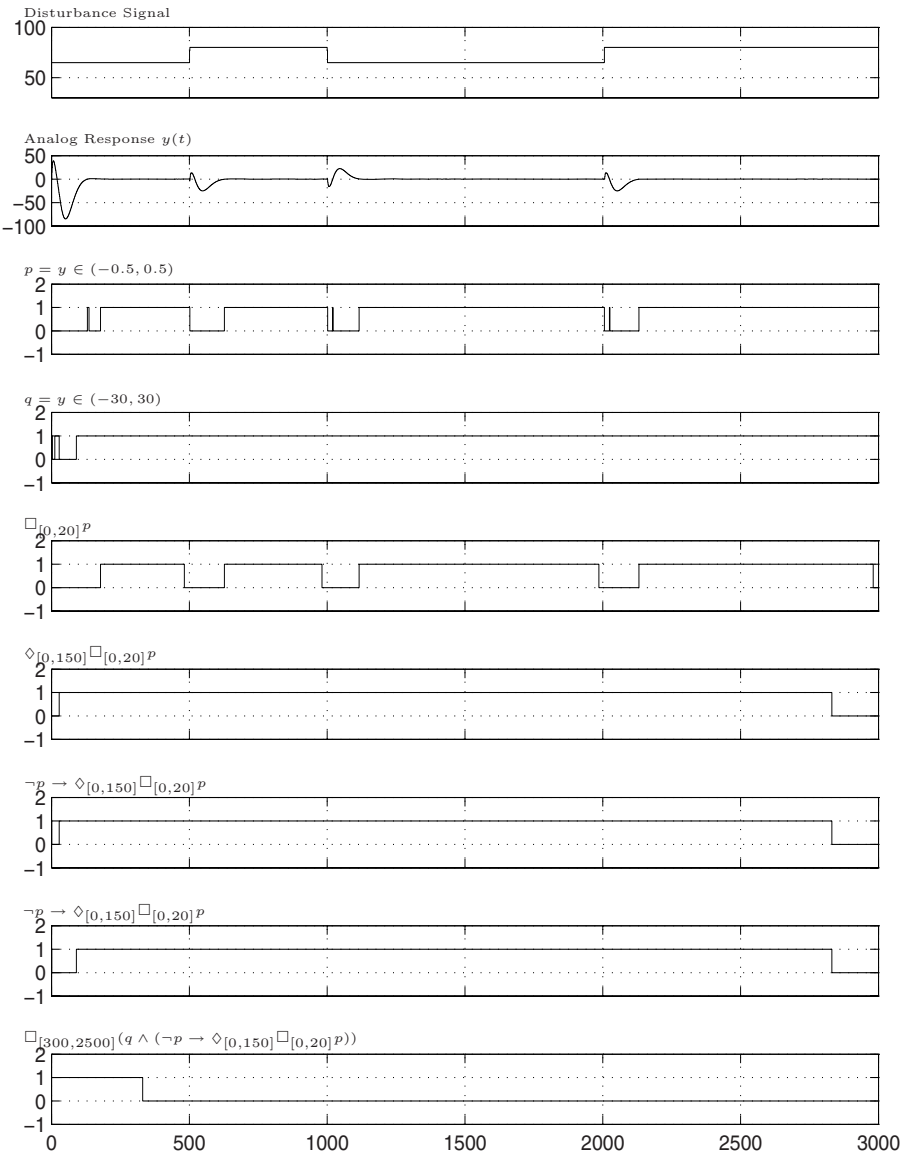


Fig. 13. A disturbance signal and an analog response y satisfying the stabilizability property $\Box_{[300,2500]}((|y| \leq 30) \wedge ((|y| > 0.5) \Rightarrow \Diamond_{[0,150]}\Box_{[0,20]}(|y| \leq 0.5)))$

system load that trigger changes in the operations of the reactor which, in turn, influences the water level, see [9]. Other instances of the same type of problem may occur when the voltage of a circuit has to be kept constant despite variations in the current due to changes in the circuit workload.

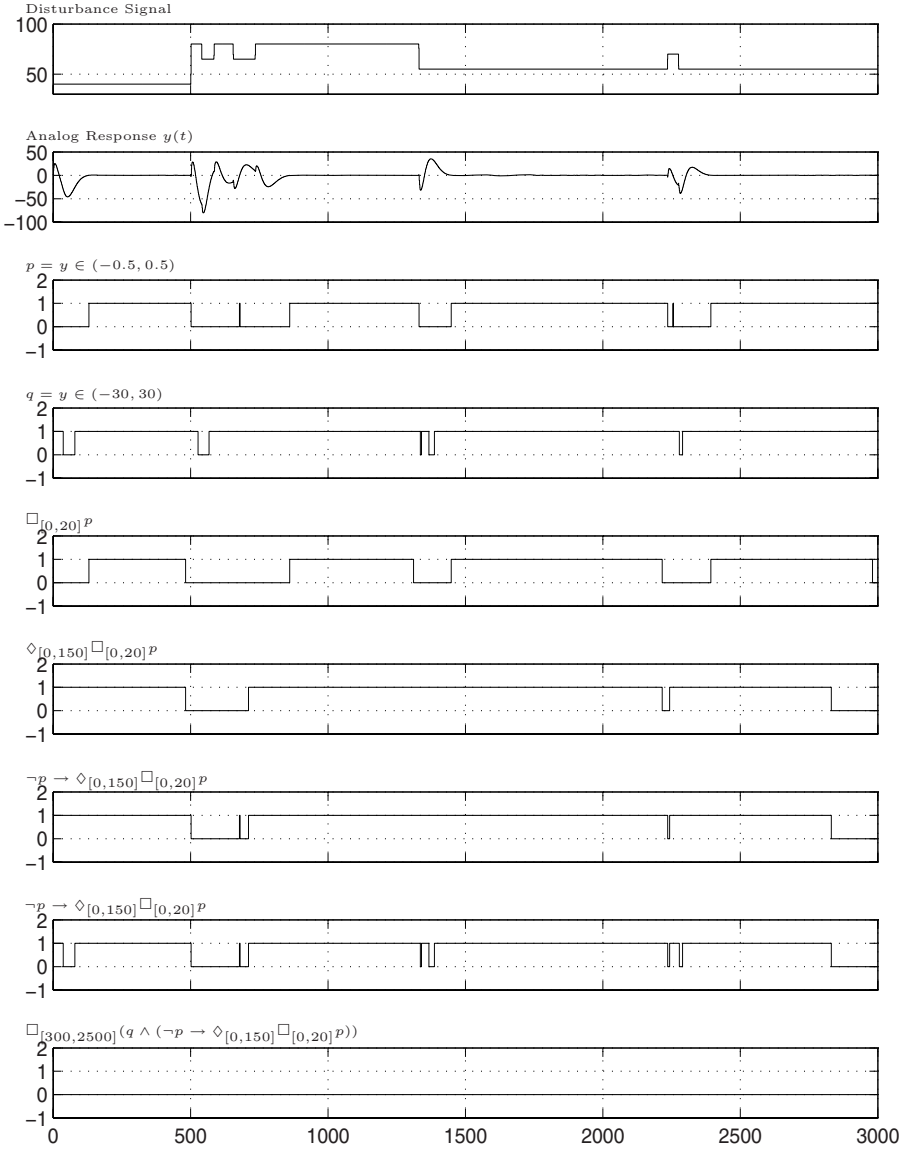


Fig. 14. A disturbance signal and an analog response y violating the stabilizability property $\Box_{[300,2500]}((|y| \leq 30) \wedge ((|y| > 0.5) \Rightarrow \Diamond_{[0,150]}\Box_{[0,20]}(|y| \leq 0.5)))$

We want y to stay always in the interval $[-30, 30]$ (except, possibly, for an initialization period of duration 300) and if, due to a disturbance, it goes outside the interval $[-0.5, 0.5]$, it should return to it within 150 time units and stay there for at least 20 time units. The whole property is

$$\varphi_2 : \quad \Box_{[300,2500]}((|y| \leq 30) \wedge ((|y| > 0.5) \Rightarrow \Diamond_{[0,150]}\Box_{[0,20]}(|y| \leq 0.5))).$$

The results of applying our offline monitoring procedure to this formula appear in Figures 13 and 14. When the disturbance is well-behaving, the property is verified, while when the disturbance changes too fast, the property is violated both by over-shooting below -30 and by taking more than 150 time units to return to $[-0.5, 0.5]$.

9 Conclusions

Motivated by the exportation of some ingredients of formal verification technology toward analog circuits and continuous systems in general, we embarked on the development of a monitoring procedure for temporal properties of continuous signals. During the process we have gained better understanding of temporal satisfiability in general as well as of the relation between real-time temporal logics and timed automata. The ideas presented in this paper have been implemented into an *analog monitoring tool* AMT [37] that has been applied to real-life case studies.

References

1. Abarbanel, Y., et al.: FoCs: Automatic Generation of Simulation Checkers from Formal Specifications. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 538–542. Springer, Heidelberg (2000)
2. Alur, R., Dill, D.L.: A Theory of Timed Automata. Theoretical Computer Science 126, 183–235 (1994)
3. Alur, R., Feder, T., Henzinger, T.A.: The Benefits of Relaxing Punctuality. Journal of the ACM 43, 116–146 (1996)
4. Alur, R., Henzinger, T.A.: Logics and Models of Real-Time: A Survey. In: Huizing, C., et al. (eds.) REX 1991. LNCS, vol. 600, pp. 74–106. Springer, Heidelberg (1992)
5. Asarin, E.: Challenges in Timed Languages. Bulletin of EATCS 83 (2004)
6. Asarin, E., Caspi, P., Maler, O.: Timed Regular Expressions. The Journal of the ACM 49, 172–206 (2002)
7. Beer, I., et al.: The Temporal Logic Sugar. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, Springer, Heidelberg (2001)
8. Bensalem, S., et al.: Testing Conformance of Real-time Applications with Automatic Generation of Observers. In: RV 2004 (2004)
9. Donzé, A.: Etude d'un Modèle de Contrôleur Hybride. Master's thesis, INPG (2003)
10. Eisner, C., Fisman, D.: A Practical Introduction to PSL. Springer, Heidelberg (2006)
11. Eisner, C., et al.: Reasoning with Temporal Logic on Truncated Paths. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 27–39. Springer, Heidelberg (2003)

12. Geilen, M.C.W., Dams, D.R.: An On-the-fly Tableau Construction for a Real-time Temporal Logic. In: Joseph, M. (ed.) FTRTFT 2000. LNCS, vol. 1926, pp. 276–290. Springer, Heidelberg (2000)
13. Geilen, M.C.W.: Formal Techniques for Verification of Complex Real-time Systems, PhD thesis, Eindhoven University of Technology (2002)
14. Geilen, M.C.W.: An Improved On-the-fly Tableau Construction for a Real-time Temporal Logic. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 394–406. Springer, Heidelberg (2003)
15. Havelund, K., Rosu, G. (eds.): Runtime Verification RV 2002. ENTCS 70(4) (2002)
16. Havelund, K., Rosu, G.: Synthesizing Monitors for Safety Properties. In: Katoen, J.-P., Stevens, P. (eds.) ETAPS 2002 and TACAS 2002. LNCS, vol. 2280, pp. 342–356. Springer, Heidelberg (2002)
17. Henzinger, T.A.: It's about Time: Real-time Logics Reviewed. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 439–454. Springer, Heidelberg (1998)
18. Hirshfeld, Y., Rabinovich, A.: Logics for Real Time: Decidability and Complexity. Fundamenta Informaticae 62, 1–28 (2004)
19. Kesten, Y., Pnueli, A.: A Compositional Approach to CTL* Verification. Theoretical Computer Science 331, 397–428 (2005)
20. Kossentini C., Caspi, P.: Approximation, Sampling and Voting in Hybrid Computing Systems, HSCC (to appear, 2006)
21. Koymans, R.: Specifying Real-time Properties with with Metric Temporal Logic. Real-time Systems, 255–299 (1990)
22. Kim, M., et al.: Monitoring, Checking, and Steering of Real-time Systems. RV 2002, ENTCS 70(4) (2002)
23. Kristoffersen, K.J., Pedersen, C., Andersen, H.R.: Runtime Verification of Timed LTL using Disjunctive Normalized Equation Systems. RV 2003 ENTCS 89(2) (2003)
24. Krichen, M., Tripakis, S.: Black-box Conformance Testing for Real-time Systems. In: Graf, S., Mounier, L. (eds.) SPIN 2004. LNCS, vol. 2989, pp. 109–126. Springer, Heidelberg (2004)
25. Kupferman, O., Vardi, M.Y.: On Bounded Specifications. In: Nieuwenhuis, R., Voronkov, A. (eds.) LPAR 2001. LNCS (LNAI), vol. 2250, pp. 24–38. Springer, Heidelberg (2001)
26. Mosterman, P.J.: An Overview of Hybrid Simulation Phenomena and their Support by Simulation Packages. In: Vaandrager, F.W., van Schuppen, J.H. (eds.) HSCC 1999. LNCS, vol. 1569, pp. 165–177. Springer, Heidelberg (1999)
27. Lichtenstein, O., Pnueli, A., Zuck, L.D.: The Glory of the Past. In: LCTES 2000. LNCS, pp. 196–218 (1985)
28. Maler, O.: A Unified Approach for Studying Discrete and Continuous Dynamical Systems. In: CDC, pp. 2083–2088. IEEE, Los Alamitos (1998)
29. Maler, O.: Control from Computer Science. Annual Reviews in Control 26, 175–187 (2002)
30. Maler, O.: Analog Circuit Verification: a State of an Art. ENTCS 153, 3–7 (2006)
31. Maler, O., Nickovic, D.: Monitoring Temporal Properties of Continuous Signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS 2004 and FTRTFT 2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004)
32. Maler, O., Nickovic, D., Pnueli, A.: Real Time Temporal Logic: Past, Present, Future. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 2–16. Springer, Heidelberg (2005)

33. Maler, O., Nickovic, D., Pnueli, A.: From MITL to Timed Automata. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 274–289. Springer, Heidelberg (2006)
34. Maler, O., Nickovic, D., Pnueli, A.: On Synthesizing Controllers from Bounded-Response Properties. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 95–107. Springer, Heidelberg (2007)
35. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems - Specification. Springer, Heidelberg (1992)
36. Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems: Safety. Springer, Heidelberg (1995)
37. Nickovic, D., Maler, O.: AMT: A Property-Based Monitoring Tool for Analog Systems. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 304–319. Springer, Heidelberg (2007)
38. Safra, S.: On the Complexity of ω-Automata. In: FOCS 1988, pp. 319–327 (1988)
39. Sokolsky, O., Viswanathan, M.(eds.): Runtime Verification RV 2003. ENTCS 89(2) (2003)
40. Trakhtenbrot, B.A.: Finite Automata and the Logic of One-place Predicates. DAN SSSR 140 (1961)
41. Thati, P., Rosu, G.: Monitoring Algorithms for Metric Temporal Logic Specifications. RV (2004)
42. Tripakis, S.: Fault Diagnosis for Timed Automata. In: Damm, W., Olderog, E.-R. (eds.) FTRTFT 2002. LNCS, vol. 2469, pp. 205–224. Springer, Heidelberg (2002)
43. Vardi, M.Y., Wolper, P.: An Automata-theoretic Approach to Automatic Program Verification. In: LICS 1986, pp. 322–331. IEEE, Los Alamitos (1986)