



FP6-IST-507219

PROSYD:

Property-Based System Design

Instrument: Specific Targeted Research Project

Thematic Priority: Information Society Technologies

Checking Digital, Timed and Analog PSL Properties (Deliverable 3.2/6)

Due date of deliverable: January 31, 2006

Actual submission date: February 7, 2005

Start date of project: January 1, 2004

Duration: Three years

Organisation name of lead contractor for this deliverable: Verimag

Revision 1.0

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	<input checked="" type="checkbox"/>
PP	Restricted to other programme participants (including the Commission Services)	<input type="checkbox"/>
RE	Restricted to a group specified by the consortium (including the Commission Services)	<input type="checkbox"/>
CO	Confidential, only for members of the consortium (including the Commission Services)	<input type="checkbox"/>

Notices

For information, contact Oded Maler maler@imag.fr.

This document is intended to fulfil the contractual obligations of the PROSYD project concerning deliverable 3.2/6 described in contract number 507219.

The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

© Copyright PROSYD 2005. All rights reserved.

Table of Revisions

Version	Date	Description and reason	By	Affected sections
0.1	January 10, 2006	Creation	Maler	
0.2	January 24, 2006	Revision	Maler	Partner feedback
0.3	January 31, 2006	Revision	Maler	Partner feedback
1.0	February 7, 2006	Revision	Maler	Polishing the format

Authors

Oded Maler (VERIMAG)
Dejan Nickovic (VERIMAG)
Amir Pnueli (Weizmann)

With contributions by Paul Caspi, Alexandre Donze, Stavros Tripakis (VERIMAG),
Cindy Eisner (IBM) and Andrea Fedeli (ST).

Executive Summary

This report explores different approaches for extending property checking algorithms toward timed and analog signals.

Purpose

The purpose of this document is to survey different approaches to the problem checking whether individual simulation traces satisfy temporal properties, and to introduce new results on the topic obtained by the PROSYD consortium for timed and analog properties.

Intended Audience

This guide is intended for practitioners, researchers and tool developers interested in the problem.

Background

The first part of the report presents in a synthetic way work that has been performed before the project in the context of hardware as well as software, with the goal of facilitating the understanding of the other parts. The rest of the report, dedicated to timed and analog properties, presents genuine PROSYD contributions at various levels of maturity.

Contents

Table of Revisions	iii
Authors	iii
Executive Summary	iii
Purpose	iii
Intended Audience	iii
Background	iii
Contents	v
Table of Figures	vii
List of Tables	ix
Glossary	x
1 Introduction	1
2 Discrete (Digital) Systems: Properties.....	3
2.1 Temporal Logic (Future)	3
2.2 Temporal Logic (Past).....	5
2.3 Regular Expressions	5
3 Discrete Systems: Checking Temporal Properties.....	7
3.1 Causality and Non-determinism	7
3.2 Evaluating Incomplete Behaviors	8
3.3 Offline and Online Monitoring	10
The Automaton-Based Method	12
Purely-Offline Marking.....	13
Incremental Marking	13
4 The Timed Level of Abstraction	15
4.1 Dense-Time Signals: Representation	16
4.2 Dense-Time Signals: Properties	17
5 Boolean Signals and their Temporal Logics	19
5.1 Signals.....	19
5.2 Real-time Temporal Logic.....	21
6 Checking Timed Properties.....	23
6.1 Offline Marking	23
6.2 Incremental Marking	25
7 Monitoring using Timed Automata	29
7.1 Timed Automata	29
7.2 From Past MITL to Deterministic Timed Automata.....	30
7.3 From Future MITL to Timed Automata.....	34
8 Analog Signals.....	39

9	Monitoring STL Properties	43
9.1	Following a Reference Signal	43
9.2	Stabilizability	44
10	Further Directions	49
10.1	The FLASH-Memory Case Study	49
10.2	Defining and Checking Metrical Properties	50
11	References.....	53

Table of Figures

Figure 1 - Offline, passive online and active online modes of interaction between a test generator and a checker.	12
Figure 2 - A 2-dimensional analog signal and the 2-dimensional Boolean signal obtained from it via the predicates $x_1 > 0.7$ and $x_2 > 0.7$	16
Figure 3 - A signal ξ and its unitary decomposition (ξ^1, ξ^2, ξ^3)	21
Figure 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 $n - a < 0$	21
Figure 5 - To compute $p \vee q$ we first refine the interval covering to obtain the a representation of the signals by p' and q' , then perform interval-wise operations to obtain $p' \vee q'$ and then merge adjacent positive intervals.	25
Figure 6 - Marking for $p\mathcal{M}_{[a,b]}q$ via marking for $\diamond_{[a,b]}(\pi \wedge p) \wedge p$: (a) with non-unitary signals we obtain wrong results; (b) with a unitary decomposition of p and q we obtain correct results. The computation with p_1 is omitted as it has an empty intersection with q	26
Figure 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.	27
Figure 8 - Memorizing changes in the truth value of φ : (a) $x_2 - y_1 \geq b - a$; (b) $x_2 - y_1 < b - a$	32
Figure 9 - An $[a, b]$ event recorder. The input labels and staying conditions are written on the bottom of each state. Transitions are decorated by the input labels of the target states and by clock resets. The clock shift operator is denoted by the symbol s	33
Figure 10 - The automaton for $\varphi \mathcal{S}_{[a,b]} \psi$	34
Figure 11 - A signal u satisfying $\square(u \equiv p\mathcal{M}_{[a,b]}q)$	35
Figure 12 - The architecture of the tester.	36
Figure 13 - The prediction generator.	37
Figure 14 - A “filter” for eliminating predictions violating Lemma 1. ...	37
Figure 15 - A negative tester \mathcal{A}_i^0	37
Figure 16 - A positive tester \mathcal{A}_i^1	38
Figure 17 - A tester for $\square(u \equiv p\mathcal{M}_{[0,b]}q)$	38

Figure 18 - Two signals which are close from a continuous point of view, one satisfying the property $\Box(x > 0)$ and one violating it.	40
Figure 19 - Shifting the sampling points, zero crossing can be missed. .	41
Figure 20 - 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.	42
Figure 21 - 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.	44
Figure 22 - A 2-dimensional signal violating the property $\Box_{[0,300]}((x_1 > 0.7) \Rightarrow \Diamond_{[3,5]}(x_2 > 0.7))$	45
Figure 23 - A disturbance signal and an analog response y satisfying the stabilizability property $\Box_{[300,2500]}((y \leq 30) \wedge ((y > 0.5) \Rightarrow \Diamond_{[0,150]}\Box_{[0,20]}(y \leq 0.5)))$	47
Figure 24 - A disturbance signal and an analog response y violating the stabilizability property $\Box_{[300,2500]}((y \leq 30) \wedge ((y > 0.5) \Rightarrow \Diamond_{[0,150]}\Box_{[0,20]}(y \leq 0.5)))$	48

List of Tables

Table 1 - The effect of the clock shifting operation while taking a transition from $(01)^i$ to $(01)^{i-1}$	32
Table 2 - CPU time of monitoring the water level controller example as a function of the time horizon (signal length). The number of positive intervals in the Boolean abstractions is given as another indication for the complexity of the problem.....	46

Glossary

Behavior A function from a time domain to a data domain. For discrete systems a behavior is a Boolean sequence; for timed system it is a Boolean signal; for analog system a behavior is a real-valued signal.

Checking The activity of verifying whether an *individual* behavior satisfies a given property. Other synonyms: monitoring, testing, runtime verification, observing, “dynamic” verification.

LTL Linear-time temporal logic.

MITL Metric interval temporal logic.

1 Introduction

This report is concerned with the following problem. Given a desired property φ of a system S , how to check that a *given* behavior ξ of S does satisfy the property φ . Within most of this report we assume that the behavior to be checked is produced by a *model*, rather than a physical realization of S , a model used during the design phase for simulation and validation.¹ While “static” verification is aimed at showing that *all* behaviors generated by the model S satisfy φ , in “dynamic” verification, S is used to generate *one behavior at a time* and can thus be viewed as a *black box*. In this framework, test-generation issues such as coverage, are delegated outside the scope of the property monitor. While the advantage of this approach is evident for system models too large to be verified statically (and, of course, for analog models used by numerical simulation tools which are sometimes hardly formalizable), the explicit presentation of ξ itself, rather than using the generating model S , raises new problems, some of which have been studied extensively in recent years both in the context of digital hardware as well as software, where monitoring is referred to as *runtime verification*.

Since the report is concerned with three different levels of abstraction (discrete, timed and analog), we start with a generic model of a dynamical system defined over an abstract state space which evolves in an abstract time domain. All other models are obtained as special instances.

States and Behaviors: A model S of a system is defined over a set $V = \{v_1, \dots, v_n\}$ of *state variables* each ranging over a domain X_i . The *state space* of the system is thus $X = X_1 \times \dots \times X_n$. 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, $\xi : T \rightarrow X$. 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 $\xi[t] = \perp$ 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 as a function of the current state and current input $u \in U$. As mentioned earlier, we do not have access to f and our interaction with

¹The relevance to monitoring and testing of *real* physical devices will be mentioned briefly.

²For 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 analog signals.

the model is restricted to stimulating it with an input $\nu \in U^*$ and then observing and checking the generated behavior ξ .

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 report 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” digital systems, whose behaviors are *continuous-time Boolean signals*, which raise a lot of new issues such as sampling, event detection, variability bounds, etc. Most of the report will investigate monitoring at this level of abstraction where we made a significant progress in the first two years of the project. Finally we move to analog signals which, in addition to dense time, admit also *numerical real values*. Although for many types of properties (and in particular those suggested in Deliverable D1.3/1, [M05]) checking analog properties can be reduced to checking timed properties, there are further issues, such as approximation errors, raised by the continuous domain and by the way signals are generated by numerical simulators.

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) encodable 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 sequence (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. We focus here on two classical formalisms, namely *temporal logic* and *regular expressions*, both included in PSL.

It should be noted that unlike further sections that deal with properties of timed and analog signal, 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 analog extensions.

³We 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.

⁴In 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.

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 $v_1 = 1$ at position t then $v_2 = 0$ at position $t + 3$. A property monitor is thus a device that observes sequences and checks whether they satisfy all such relations. 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 $P = \{p_1, \dots, p_n\}$ of propositions indicating values of the corresponding state variable. The basic temporal operators are *next* (\bigcirc), which specifies what should hold in the next step⁵ and the *until* operator (\mathcal{U}), which requires φ_1 to hold until φ_2 becomes true, without bounding the temporal distance to this becoming. From basic LTL operators one can derive other standard Boolean operators like \top and \perp for *true* and *false* and temporal operators such as *eventually* (\diamond) and *always* (\square):

$$\diamond\varphi = \top \mathcal{U} \varphi \quad \text{and} \quad \square\varphi = \neg \diamond \neg \varphi.$$

Models of LTL are *Boolean sequences* of the form $\xi : \mathbb{N} \rightarrow \mathbb{B}^n$. 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:⁶

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

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

⁵And which becomes meaningless when the time domain is dense.

⁶Due to their importance we also give explicitly the semantics of the derived temporal operators *always* and *eventually*.

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* \square defined as

$$\diamond\varphi = \top \mathcal{S}\varphi \quad \text{and} \quad \square\varphi = \neg\diamond\neg\varphi$$

Their semantics is given by

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

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

2.3 Regular Expressions

Regular expressions constitute a well-established formalism for describing sets of sequences which has some similarities and some differences to temporal logic. One difference is that traditionally temporal logic is used to define ω -languages (sets of infinite sequences) while regular expressions are used mainly for finite sequences.⁷ The second difference is that temporal logic is better suited for handling multi-dimensional sequences while regular expressions are traditionally oriented toward “monolithic” alphabets. In other words, a temporal logic formula over a set $\{p_1, \dots, p_n\}$ of propositional variables defines a language over an alphabet $\Sigma = \mathbb{B}^n$ where every “letter” $\sigma \in \Sigma$ is a Boolean vector. In this framework, a proposition p_1 in a regular expression over \mathbb{B}^2 is a shorthand for $(1, 0) \vee (1, 1)$. Finally the major temporal operator is the concatenation operation which differs from the *until* and has a more “synchronous” nature.

⁷There are also ω -regular expressions, though.

The standard syntax of regular expressions over an alphabet Σ is given by the grammar

$$\varphi := \emptyset | \varepsilon | \sigma | \varphi_1 \vee \varphi_2 | \varphi_1 \cdot \varphi_2 | \varphi^* .$$

The satisfaction of a regular expression by a sequence ξ defined over $[0..r - 1]$ is given by the following semantic rules.

$$\begin{aligned} \xi \models \emptyset &\leftrightarrow \text{false} \\ \xi \models \varepsilon &\leftrightarrow \xi[0] = \perp \\ \xi \models \sigma &\leftrightarrow \xi[0] = \sigma \\ \xi \models \varphi_1 \vee \varphi_2 &\leftrightarrow \xi \models \varphi_1 \text{ or } \xi \models \varphi_2 \\ \xi \models \varphi_1 \cdot \varphi_2 &\leftrightarrow \exists k \xi[0..k - 1] \models \varphi_1 \text{ and } \xi[k..r - 1] \models \varphi_2 \\ \xi \models \varphi^* &\leftrightarrow \exists n \geq 0 \exists k_1 < k_2 < \dots < k_n = r \forall i \xi[k_i, k_{i+1} - 1] \models \varphi \end{aligned}$$

This syntax defines the regular (rational) languages. Adding negation and complementation does not add to the expressive power but may lead to more succinct representation (and hence to more complex satisfaction test). The major difficulty here is the *concatenation* operator because to check whether ξ satisfies $\varphi_1 \cdot \varphi_2$ one needs to find (guess) an appropriate *factorization* of ξ . The *Kleene star* operator used to concatenate an arbitrary but finitely-many non-empty segments satisfying φ can be augmented with the ω -*exponentiation* operator with φ^ω indicating any infinite sequence that can be obtained by concatenating infinitely-many non-empty segments that satisfy φ . With this syntax one obtains the ω -regular sets. The expressive power of regular expressions is strictly larger than that of LTL which is matched by another class of expressions called the *star-free* regular expressions, defined by the syntax

$$\varphi := \emptyset | \varepsilon | \sigma | \varphi_1 \vee \varphi_2 | \varphi_1 \cdot \varphi_2 | \neg \varphi .$$

The PSL language uses two additional constructs the combine regular expressions and temporal logic, namely $\langle \varphi \rangle \psi$ and $[\varphi] \psi$ whose semantics are defined as

$$\begin{aligned} \xi \models \langle \varphi \rangle \psi &\leftrightarrow \exists k \xi[0..k - 1] \models \varphi \text{ and } \xi[k..r - 1] \models \psi \\ \xi \models [\varphi] \psi &\leftrightarrow \forall k \xi[0..k - 1] \models \varphi \text{ implies } \xi[k..r - 1] \models \psi \end{aligned}$$

As one can see $\langle \varphi \rangle \psi$ is simply a concatenation of sets defined by two different formalisms, while $[\varphi] \psi$, also known as *sequence implication*, requires that $\xi_2 \models \psi$. for any factorization $\xi = \xi_1 \cdot \xi_2$ such that $\xi_1 \models \varphi$.

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 analog 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 initially called with formula 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 $t' > t$. Even worse, some temporal operators refer to future time instants in a *quantified* manner, for example, requiring some p to hold in *all time instants*. The satisfiability of such a property may sometime be determined only at infinity, that is, “after” we can be sure that no $\neg p$ will be 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 appropriate for humans. The past fragment of LTL admits an immediate translation to deterministic automata and a simple monitoring procedure [HR02b] based on this observation.

The “classical” theoretical scheme for using LTL in static verification is based on translating a formula φ into a non-deterministic ω -automaton⁸ \mathcal{A}_φ that accepts exactly the sequences that satisfy it. The non-determinism is needed to compensate

⁸That is, an automaton over infinite sequences.

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}_φ is not empty. This reduces to checking the existence of an accepting cycle in \mathcal{A}_φ which is reachable from an initial state. Static verification (“model checking”) is achieved by checking whether S may generate an infinite behavior rejected by \mathcal{A}_φ (or accepted by $\mathcal{A}_{\neg\varphi}$). It should be noted that simplified procedures have been developed and implemented when the property in question belong 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.⁹ 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*.

⁹To 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 and additional information 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.

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 $\bigcirc^k p$, 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

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

Positive and negative determination correspond, respectively, to the special cases where $\xi \setminus \varphi = X^*$ and $\xi \setminus \varphi = \emptyset$. 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 [LPZ85, EFH⁺03]. 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 signal up to some $t \leq k$, with k being a constant depending on φ . Note that this class is 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 [KV01]).

The next class is the class of *safety* properties¹⁰ 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

¹⁰To 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.

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 $\Box p$. As soon as $\neg p$ is observed, we can say the the formula is negatively determined and need not wait for the end 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 and the fact that their corresponding languages are prefix-closed, like the languages that correspond to sequences that do not negatively determine safety properties.

Naturally many solutions have been proposed to this problem in the context of monitoring and runtime verification and we mention few. The work of [ABG⁺00] concerning the FoCs property checker of IBM, as well as those of [KLS⁺02] 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 [KPA03] and [TR04] in the context of timed properties. The most systematic study of adapting LTL semantics to finite sequences (“truncated paths”) is presented in [EFH⁺03]. This semantics has been adopted for PSL.

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 $\Omega_\varphi : X^* \rightarrow D$ which maps all finite sequences into a domain D that contains \mathbb{B} (satisfied/violated) and is augmented with some additional values for undecided formulae.

The next thing to do is to describe the different ways to embed the process of computing $\Omega_\varphi(\xi)$ with the dynamic process of generating ξ .

3.3 Offline and Online Monitoring

The question of online vs. offline monitoring is concerned with the 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*¹¹ and, hence, the only natural way to generate behaviors is from the past to the future. One may think of three basic modes of interaction (see also Figure 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 tester 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 of violation, and is outside the scope of this report.

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, we 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 in 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 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.

¹¹A dynamical system is reverse-deterministic if it is deterministic when we reverse the direction. Only permutation automata and purely-continuous dynamical systems are reverse deterministic.

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

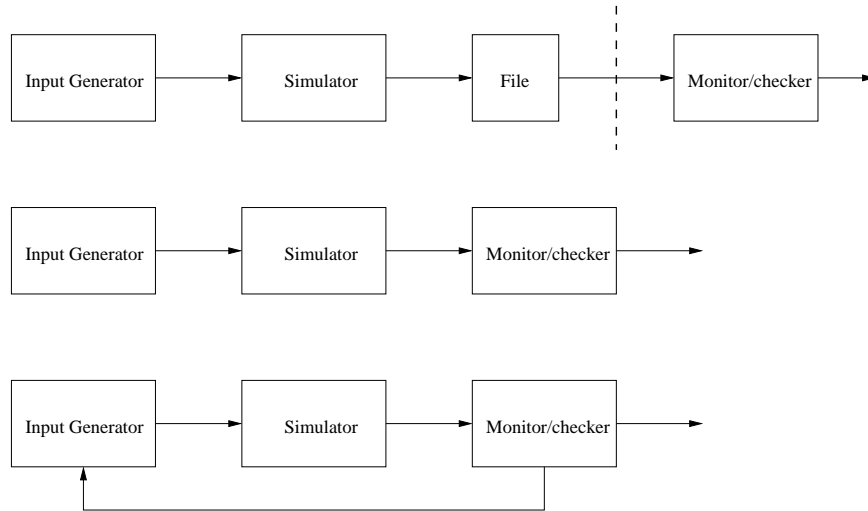


Figure 1: Offline, passive online and active online modes of interaction between a test generator and a checker.

The Automaton-Based Method

This is an online-oriented approach that follows the principles used in static verification. To monitor a property φ we first construct the automaton \mathcal{A}_φ 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 [VW86] and our approach to monitoring does not depend on the choice of the translation algorithm. We have, however a preference for the compositional construction, based on [KP05]. 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 if $(\xi, t) \models \psi$.

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}_φ 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 [S88] or using a simpler algorithm adapted to the finitary semantics.

Purely-Offline Marking

This is the first method we have applied to timed and analog properties and will be described in more detail in Section 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 a more algorithmic detail in Section 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, we face two major conceptual and technical problems.

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.

Mathematicall 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 checking 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 employ the following strategy. As a first version of the analog extensions of PSL, we use the logic STL (*signal temporal logic*, see deliverable D1.3/1 [M05]) which is based on transforming analog signals into Boolean ones and concentrate most of our efforts toward checking properties of those dense-time Boolean signals, expressed in MITL (metric interval temporal logic). This allows us to tackle the problem of dense time in isolation. Aspects specific to the continuous state space are discussed in Section 8.

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 $\{p_1, \dots, p_n\}$ 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 μ_1, \dots, μ_m be m predicates of the form $\mu : \mathbb{R}^n \rightarrow \mathbb{B}$. These predicates define a mapping $M : \mathbb{R}^n \rightarrow \mathbb{B}^m$ assigning to every real point a Boolean vector indicating the predicates it satisfies. Applying this mapping in a pointwise fashion to an analog signal $\xi : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^n$ we obtain a Boolean signal $= M(\xi) = \xi' : \mathbb{R}_{\geq 0} \rightarrow \mathbb{B}^m$ describing the evolution over time of the truth values of these predicates with respect to ξ (see Figure 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.

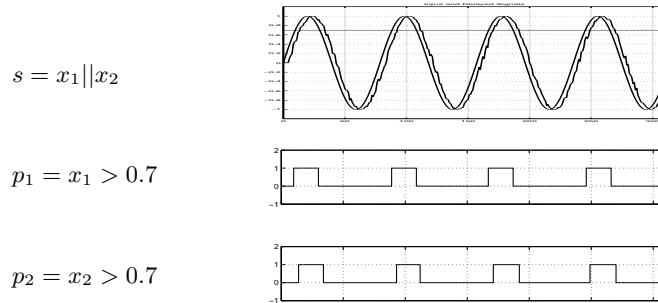


Figure 2: A 2-dimensional analog signal and the 2-dimensional Boolean signal obtained from it via the predicates $x_1 > 0.7$ and $x_2 > 0.7$.

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 $t' = \text{Suc}^k(t)$. 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 $\xi[0], \xi[1], \dots, \xi[r]$.

The dense order $(\mathbb{R}, <)$ does not admit such a property, and for every $t < t'$ one can find a 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 Section 8). 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).¹² 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, \dots, 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 .

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 underspecified 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 least 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 [Koy90, AH92, Hen98, HR04] and we will focus on the logic MITL, which is a natural adaptation of LTL to dense time [AFH96].

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 sub-formulae 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

¹²Such 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 $x < 0$. But this will happen only with an ideal simulator with unbounded precision, while a real simulator will eventually settle in some value.

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 Section 7.3 we will mention a remedy to this problem.

We can now move to more detailed definitions of signals and their corresponding temporal logics, followed by the description of the various monitoring algorithms that we have developed for them.

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 [ACM02] 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 $\xi : T \rightarrow \mathbb{B}^n$. 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 length r (denoted by $|\xi| = r$). We use the notation $\xi[t] = \perp$ when $t \geq |\xi|$. The restriction of a signal to length d is defined as

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

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 [ACM02, A04].

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

$$\begin{aligned} \xi_1 \parallel \xi_2 = \xi_{12} & \text{ if } \forall t \xi_{12}[t] = (\xi_1[t], \xi_2[t]) \\ \xi_1 = \pi_1(\xi_{12}) \quad \xi_2 = \pi_2(\xi_{12}) \end{aligned}$$

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 $\mathcal{I} = I_1, I_2 \dots$ 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}' satisfying $\mathcal{I}' \prec \mathcal{I}$. We restrict ourselves to signals of *finite variability*, that is, signals admitting a finite consistent interval covering. We denote by \mathcal{I}_ξ 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}_ξ^+ is a singleton. Any finite-variability signal ξ can be decomposed into a union of k unitary signals such that $\xi = \xi^1 \vee \dots \vee \xi^k$, see Figure 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:

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

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

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

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

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

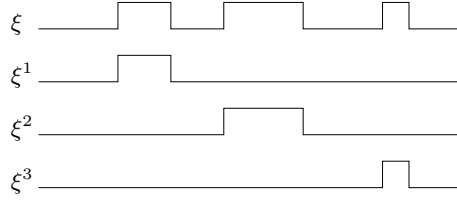


Figure 3: A signal ξ and its unitary decomposition (ξ^1, ξ^2, ξ^3) .

Of particular interest are the applications of these operational to one-dimensional sets consisting of elements of the time domain \mathbb{T} :

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

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

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

Definition 1 (Back Shifting) *The $[a, b]$ -back-shifting of a Boolean signal ξ' , denoted by $\xi = \text{SHIFT}_{[a,b]}(\xi')$, is a signal ξ such that 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 Section 5.2) is not a coincidence. If $\varphi = \diamond_{[a,b]}\varphi'$ then the respective satisfiability signals of φ and φ' satisfy $\chi_\varphi = \text{SHIFT}_{[a,b]}(\chi_{\varphi'})$. This operation is easy to compute on a representation based on an interval cover 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 $I = I' \ominus [a, b] \cap \mathbb{T}$ (the intersection with \mathbb{T} is needed to remove negative values, see Figure 4).

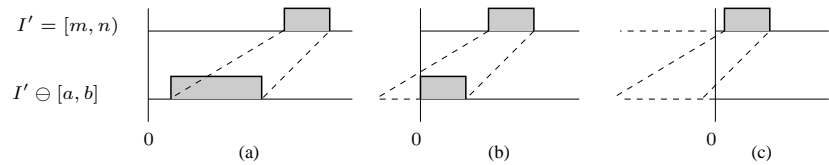


Figure 4: Three instances of back shifting $I = [m, n] \ominus [a, b]$: (a) $I = [m - b, n - a]$; (b) $I = [0, n - a]$ because $m - b < 0$; (c) $I = \emptyset$ because $n - a < 0$

5.2 Real-time Temporal Logic

The syntax of MITL is defined by the grammar

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

where p belongs to a set $P = \{p_1, \dots, p_n\}$ of propositions and $b > a \geq 0$ are rational numbers (in fact, it is sufficient to consider integer constants). From basic MITL operators one can derive other standard Boolean and temporal operators, in particular the time-constrained *eventually* and *always* operators:

$$\diamond_{[a,b]} \varphi = \top \mathcal{U}_{[a,b]} \varphi \quad \text{and} \quad \square_{[a,b]} \varphi = \neg \diamond_{[a,b]} \neg \varphi$$

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

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

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* (\square), operators. The semantics of the past operators is defined as

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

Most of the work in this section is dedicated to the more difficult future fragment of MITL.

6 Checking Timed Properties

In this section we describe three procedures for checking MITL properties that have been developed during the first two years of PROSYD. These procedure are:

1. An offline marking procedure that propagates truth values upwards from propositions via super-formlae up to the main formula. This procedure has been a subject of a scientific publication [MN04] and has been implemented and applied to some examples.
2. An incremental marking procedure that updates the marking each time a new segment of the signal is observed. This procedure has been implemented as well but has not yet been published in the scientific literature.
3. An approach based on translating the formula into a timed automaton. This is the most recent development and is the topic of a recently-submitted paper [MNP06]. The translation is useful for both static verification and monitoring where it should be combined with the on-the-fly determinization of [Tri02, KT04]. In addition we have developed an automatic translation from past MITL to deterministic timed automata [MNP05]. These automata are immediately amenable to monitoring.

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 . This signal satisfies $\xi'[t] = 1$ iff $(\xi, t) \models \varphi$. 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, developed during the first year of PROSYD [MN04] 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 long enough. 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  $\varphi$ 
switch  $\varphi$  do
  case  $p$ 
  |  $\chi_\varphi := \pi_p(\xi)$ ;
  end
  case  $OP_1(\varphi_1)$ 
  | OFFLINEMITL ( $\varphi_1$ );
  |  $\chi_\varphi := \text{COMBINE}(OP_1, \varphi_1)$ ;
  end
  case  $OP_2(\varphi_1, \varphi_2)$ 
  | OFFLINEMITL ( $\varphi_1$ );
  | OFFLINEMITL ( $\varphi_2$ );
  |  $\chi_\varphi := \text{COMBINE}(OP_2, \chi_{\varphi_1}, \chi_{\varphi_2})$ ;
  end
end

```

Most of the work in this algorithm is done by the COMBINE function which for $\varphi = OP_2(\varphi_1, \varphi_2)$ computes χ_φ from the signals χ_{φ_1} and χ_{φ_2} , which may differ 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.

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

$\chi_\varphi := \text{COMBINE}(\vee, \chi_{\varphi_1}, \chi_{\varphi_2})$ For the disjunction we first construct a refined interval covering $\mathcal{I} = \{I_1, \dots, I_k\}$ for $\chi_{\varphi_1} \parallel \chi_{\varphi_2}$ 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 $\mathcal{I}_{\chi_\varphi}$. The disjunction procedure is illustrated in Figure 5.

$\chi_\varphi := \text{COMBINE}(\diamond_{[a,b]}, \chi_{\varphi_1})$ This is the most important part of our procedure which computes $\chi_\varphi := \text{SHIFT}_{[a,b]}(\xi_{\varphi_1})$. For every positive interval $I \in$

$\mathcal{I}^+_{\varphi_1}$ we compute its back shifting $I \ominus [a, b] \cap \mathbb{T}$ and insert it to \mathcal{I}^+_{φ} . Overlapping positive intervals in \mathcal{I}^+_{φ} are merged to obtain a minimal consistent interval covering. In the process, all the negative intervals shorter than $b - a$ disappear.¹³

$\chi_{\varphi} := \mathbf{COMBINE}(\mathcal{U}_{[a,b]}, \chi_{\varphi_1}, \chi_{\varphi_2})$ 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 all the interval. This does not hold for arbitrary signals, see Figure 6. In order to treat the general case where χ_{φ_1} is an arbitrary signal we first need to decompose it into the unitary signals $\chi_{\varphi_1}^1, \dots, \chi_{\varphi_1}^k$ and then compute

$$\chi_{\varphi}^i = \mathbf{SHIFT}_{[a,b]}(\chi_{\varphi_1}^i \wedge \chi_{\varphi_2}) \wedge \chi_{\varphi_1}^i$$

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

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

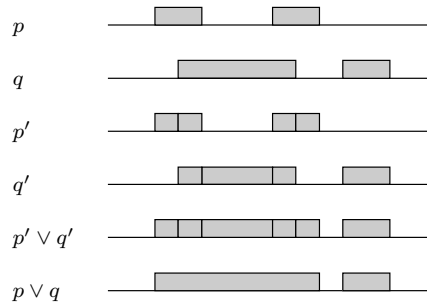
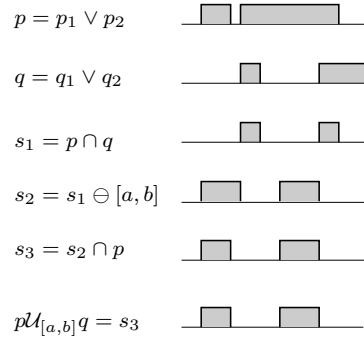


Figure 5: To compute $p \vee q$ we first refine the interval covering to obtain the a representation of the signals by p' and q' , then perform interval-wise operations to obtain $p' \vee q'$ and then merge adjacent positive intervals.

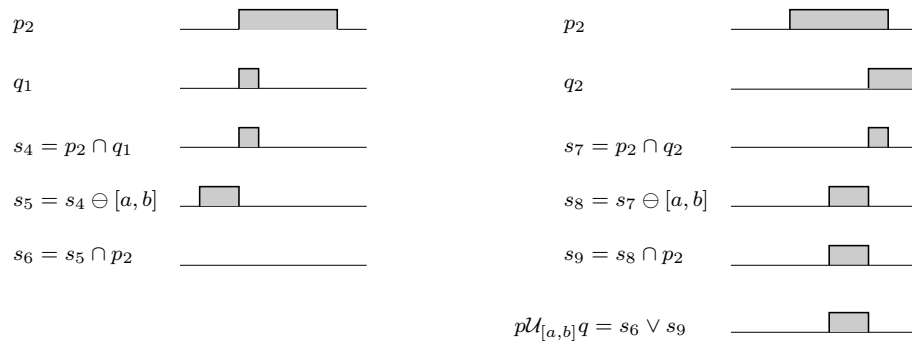
6.2 Incremental Marking

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 subformula ψ the algorithm stores its already-computed satisfying signal partitioned into a concatenation of two signals $\chi_{\psi} \cdot \Delta_{\psi}$ with χ_{ψ} consisting of values already

¹³Another way to see it is as shifting the *negative* intervals by $[b, a]$.



(a)



(b)

Figure 6: Marking for $p\mathcal{U}_{[a,b]}q$ via marking for $\diamond_{[a,b]}(\pi \wedge p) \wedge p$: (a) with non-unitary signals we obtain wrong results; (b) with a unitary decomposition of p and q we obtain correct results. The computation with p_1 is omitted as it has an empty intersection with q .

propagated to the super-formula of ψ , and Δ_ψ , consists of values that have already been computed but which have not yet propagated to the super-formula and can still influence it.

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 has 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 an illustration consider $\varphi = \text{OP}(\varphi_1, \varphi_2)$ and the corresponding truth signals of Figure 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.

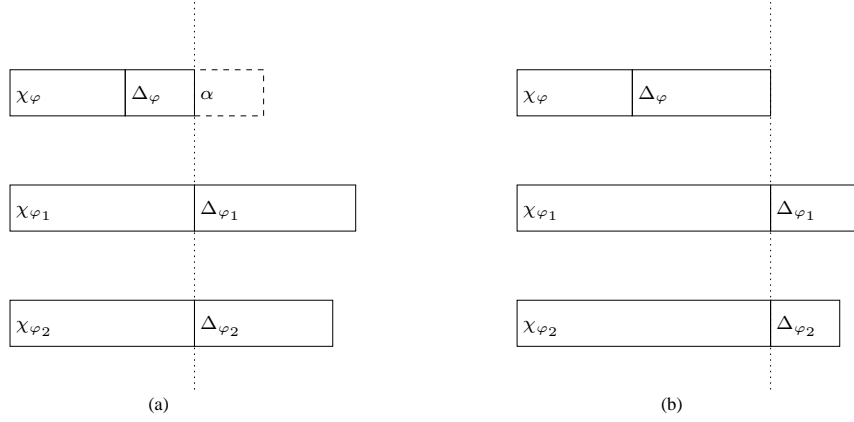


Figure 7: A step in an incremental update: (a) A new segment α for φ is computed from Δ_{φ_1} and Δ_{φ_2} ; (b) α is appended to Δ_φ and the endpoints of χ_{φ_1} and χ_{φ_2} are shifted forward accordingly.

Algorithm 2: INC-OFFLINE-MITL

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

```

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

```

7 Monitoring using Timed Automata

In this section we present several methods for checking timed properties which are based on the construction of a timed automaton \mathcal{A}_φ that accepts exactly the models of an MITL formula φ . This section presents results that were obtained during different periods during the project lifetime, some of which will be probably subsumed by the incremental algorithm described in Section 6.2. We start with a definition of timed automata, continue with our result concerning the direct translation from past MITL to *deterministic* timed automata [MNP05], followed by our newly discovered translation from future MITL to non-deterministic timed automata [MNP06] and a discussion on the applicability of the determinization procedure of [KT04] to monitoring based on this automaton. These sections are rather technical and can be skipped by readers satisfied with the procedures described in the preceding sections.

7.1 Timed Automata

We use a variant of timed automata which differs slightly from the classical definitions [AD94], [HNSY94] as it reads multi-dimensional *dense-time* Boolean signals, hence the alphabet letters are associated with *states* rather than with *transitions*. We also extend the domain of clock values to include the special symbol \perp indicating that the clock is currently *inactive*.¹⁴

The set of valuations of a set $\mathcal{C} = \{x_1, \dots, x_n\}$ of clock variables, each denoted as $v = (v_1, \dots, v_n)$, defines the clock space $\mathcal{H} = (\mathbb{R}_{\geq 0} \cup \{\perp\})^n$. A *configuration* of a timed automaton is a pair of the form (q, v) with q being a discrete state. For a clock valuation $v = (v_1, \dots, v_n)$, $v + t$ is the valuation (v'_1, \dots, v'_n) such that $v'_i = v_i$ if $v_i = \perp$ and $v'_i = v_i + t$ otherwise. A *clock constraint* is a Boolean combination of conditions of the forms $x \geq d$ or $x > d$ for some integer d .

¹⁴This is a syntactic sugar since clock inactivity in a state can be encoded implicitly by the fact that in all paths emanating from the state, the clock is reset to zero before being tested [DY96].

Definition 2 (Timed Automaton) A timed automaton over signals is a tuple $\mathcal{A} = (\Sigma, Q, \mathcal{C}, \lambda, I, \Delta, q_0, F)$ where Σ is the input alphabet (\mathbb{B}^n in this paper), Q is a finite set of discrete states and \mathcal{C} is a set of clock variables. The labeling function $\lambda : Q \rightarrow 2^\Sigma$ associates a subset of the alphabet to every state while the staying condition (invariant) I assigns to every state q a subset I_q of \mathcal{H} defined by a conjunction of inequalities of the form $x \leq d$, for some clock x and integer d . The transition relation Δ consists of elements of the form (q, g, ρ, q') where q and q' are discrete states, the transition guard g is a subset of \mathcal{H} defined by a clock constraint and ρ is the update function, a transformation of \mathcal{H} defined by assignments of the form $x := 0$ or $x := \perp$. Finally q_0 is the initial state and $F \subseteq Q$ is the acceptance condition.

The behavior of the automaton as it reads a signal ξ consists of an alternation between time progress periods where the automaton stays in a state q as long as $\xi[t] \in \lambda(q)$ and I_q holds, and discrete instantaneous transitions guarded by clock conditions. Formally, a *step* of the automaton is one of the following:

- A time step: $(q, v) \xrightarrow{\sigma^t} (q, v + t)$, $t \in \mathbb{R}_+$ such that $\sigma \in \lambda(q)$ and $v + t$ satisfies I_q (due to the structure of I_q this holds as well for every $t', 0 \leq t' < t$).
- A discrete step: $(q, v) \xrightarrow{\delta} (q', v')$, for some transition $\delta = (q, g, \rho, q') \in \Delta$, such that v satisfies g and $v' = \rho(v)$

A *run* of the automaton starting from a configuration (q_0, v_0) is a finite or infinite sequence of alternating time and discrete steps of the form

$$\xi : (q_0, v_0) \xrightarrow{\sigma_1^{t_1}} (q_0, v_0 + t_1) \xrightarrow{\delta_1} (q_1, v_1) \xrightarrow{\sigma_2^{t_2}} (q_1, v_1 + t_2) \xrightarrow{\delta_2} \dots,$$

such the $\sum t_i$ diverges. A run is accepting if the set of time instants in which it visits states in F is unbounded. The signal carried by the run is $\sigma_1^{t_1} \cdot \sigma_2^{t_2} \dots$. The language of the automaton consists of all signals carried by accepting runs.

7.2 From Past MITL to Deterministic Timed Automata

In this section we show how to build a deterministic timed automaton for any bounded past MITL formula. The construction follows the same lines as the compositional construction of [KP05, Pnu03] for untimed future temporal logic, where an automaton for a formula observes the states of the automata that correspond to

its sub-formulae. This construction is particularly attractive for past temporal logic where the correspondence between states in the automaton and satisfaction of a sub formula is more direct.

We illustrate the idea underlying the proof on the formula $\diamond_{[a,b]}\varphi$ for some past formula φ . Intuitively, an automaton that accepts such a language should monitor the truth value of φ and memorize, using clocks, the times when this value has changed. Memorizing all such changes may require an *unbounded* number of clocks, but as we shall see, only a finite number of those is sufficient since not all occurrence times of these changes need to be remembered.

Consider signal φ of Figure 8-(a), a clock x_i reset to zero at the i^{th} time φ becomes true and a clock y_i reset when φ becomes false. For this example $\diamond_{[a,b]}\varphi$ is true exactly when $(x_1 \geq a \wedge y_1 \leq b) \vee (x_2 \geq a \wedge y_2 \leq b)$. Due to the monotonicity of the clock dynamics, whenever y_1 goes beyond b , its value becomes irrelevant for the satisfaction of the acceptance condition, it can be discarded together with x_1 . By itself, this fact does not guarantee finiteness of the number of clocks because we assume no a-priori bound on the variability of φ .

Consider now Figure 8-(b), where the second rise of φ is less than $b - a$ time after the preceding fall. In this case, condition $(x_1 \geq a \wedge y_1 \leq b) \vee (x_2 \geq a \wedge y_2 \leq b)$ becomes equivalent to $x_1 \geq a \wedge y_2 \leq b$. Since the values of y_1 and x_2 do not matter anymore we may deactivate them and forget this short episode of $\neg\varphi$. When φ falls again we may re-use clock y_1 to record the occurrence time and let the acceptance condition be $x_1 \geq a \wedge y_1 \leq b$. Hence the maximal number of events to be remembered before the oldest among them expires is $m = b/(b - a) - 1$ and at most $2m$ clocks are sufficient for monitoring such a formula. Note that for a “punctual” modality where $a = b$, m goes to infinity.

The automaton depicted in Figure 9, is a kind of an “*event recorder*” for accepting signals satisfying $\diamond_{[a,b]}\varphi$. Its set of discrete states Q is partitioned into

$$Q_{\neg\varphi} = \{(01)^i 0\}_{i=0..m} \text{ and } Q_{\varphi} = \{(01)^i\}_{i=1..m},$$

with the intended meaning that the Boolean sequences that encode states correspond to the qualitative histories that they memorize, that is, the patterns of remembered rising and falling of φ that have occurred less than b time ago. The clocks of the automaton are $\{x_1, y_1, \dots, x_m, y_m\}$, each measuring the time since its corresponding event. Naturally, clock x_i is active only at states $(01)^j$ and $(01)^j 0$ for $j \geq i$ and clock y_i at $(01)^j 0$ $(01)^{j+1}$ for $j \geq i$.

When φ first occurs the automaton moves from 0 to 01 and resets x_1 . When φ becomes false it moves to 010 while resetting y_1 . From there the following three continuations are possible:

1. If φ remains false for more than b time, the true episode of φ can be forgotten and the automaton moves to 0

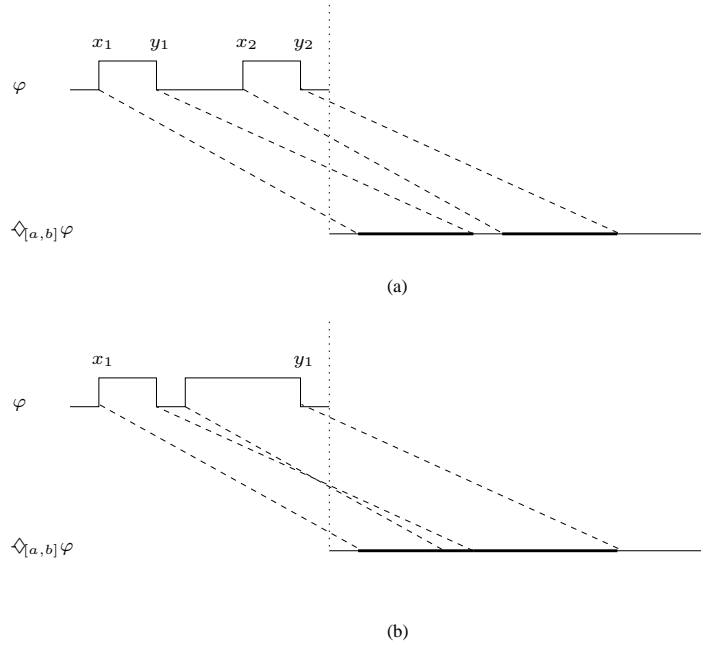


Figure 8: Memorizing changes in the truth value of φ : (a) $x_2 - y_1 \geq b - a$; (b) $x_2 - y_1 < b - a$.

2. If φ becomes true within less than $b - a$ time, the false episode is forgotten and the automaton returns to 01
3. If φ becomes true after more than $b - a$ time the automaton resets x_2 and moves to 0101.

Transitions of type 1 may happen in all states that record 2 changes or more. They occur when the first falling of φ is more than b time old and hence the values of clocks x_1 and y_1 can be forgotten. In order to keep the number of clocks bounded, this transition is accompanied by “shifting” the clocks values, that is, applying the operations $x_i := x_{i+1}$ and $y_i := y_{i+1}$ for all i as well as $x_m := y_m := \perp$. The effect of this shifting operation when a transition from $(01)^i$ to $(01)^{i-1}$ is taken is illustrated in Table 1.

c	x_1	y_1	\dots	x_{i-1}	y_{i-1}	x_i	y_i	\dots	x_m	y_m
v	u_1	v_1	\dots	u_{i-1}	v_{i-1}	u_i	\perp	\dots	\perp	\perp
$s(v)$	u_2	v_2	\dots	u_i	\perp	\perp	\perp	\dots	\perp	\perp

Table 1: The effect of the clock shifting operation while taking a transition from $(01)^i$ to $(01)^{i-1}$.

Lemma 1 *The event recorder automaton, running in parallel with the automaton \mathcal{A}_φ , accepts the signals satisfying $\diamond_{[a,b]}\varphi$ whenever x_1 is active and satisfies $x_1 \geq a$.*

Sketch of Proof: We need to show that in every state of the form $(01)^i 0$ there have been i risings and fallings of φ that have occurred less than b time ago such that each falling has lasted for more than $b - a$ time, and that the corresponding clocks represent the times elapsed since they have occurred. When this is the case and since $y_1 \leq b$ by construction, $x_1 \geq a$ at time t iff there was a time $t' \in t \ominus [a, b]$ in which φ was true. The proof is by induction on the length of the run. The claim is trivially true at the initial state. The inductive step starts with a configuration of the automaton satisfying the above, and proceeds by showing that it is preserved under time passage and transitions. The proof for states of the form $(01)^i$ is similar. ■

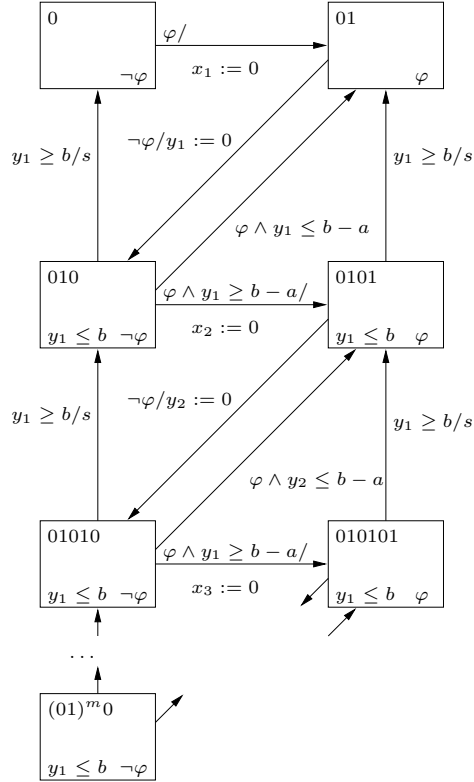


Figure 9: An $[a, b]$ event recorder. The input labels and staying conditions are written on the bottom of each state. Transitions are decorated by the input labels of the target states and by clock resets. The clock shift operator is denoted by the symbol s .

Lemma 2 Given deterministic timed automata \mathcal{A}_φ and \mathcal{A}_ψ accepting $\llbracket \varphi \rrbracket$ and $\llbracket \psi \rrbracket$, respectively, one can construct a deterministic timed automaton accepting $\varphi \mathcal{S}_{[a,b]} \psi$.

Proof: Observe first that $\varphi \mathcal{S} \psi$ can be seen as a restriction of $\diamond \psi$ to periods where φ holds *continuously*. In other words, the automaton need not measure times of changes in ψ after which φ became false. Hence the \mathcal{S} -automaton (Figure 10) consists of an event recorder for ψ augmented with an additional initial state $\neg\varphi$. Whenever φ becomes true the automaton moves to the initial state of the event

recorder and whenever φ becomes false it moves (from any state) back to $\neg\varphi$ while forgetting all the past history of ψ . \blacksquare

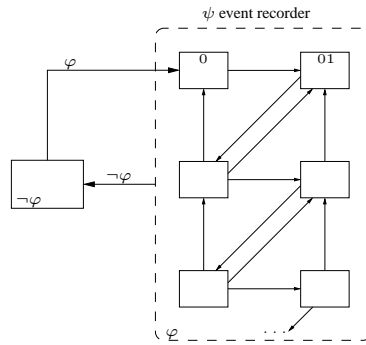


Figure 10: The automaton for $\varphi \mathcal{S}_{[a,b]} \psi$.

Theorem 3 (Past MITL is Deterministic) *Given a past MITL formula φ , one can construct a deterministic timed automaton \mathcal{A} accepting $\llbracket \varphi \rrbracket$.*

Proof: By induction on the structure of the formula. For a proposition p we build the deterministic two-state automaton \mathcal{A}_p which moves to and from the accepting state according to the current value of p . For $\neg\varphi$ we take the automaton \mathcal{A}_φ and complement its acceptance condition while for $\varphi \vee \psi$ we do a Cartesian product of \mathcal{A}_φ and \mathcal{A}_ψ . Combining this with the previous lemma the result is established. \blacksquare

This result can be readily applied for monitoring. However it is commonly accepted that future MITL is more intuitive from the user's point of view. Since another result of [MNP05] shows that future MITL is not deterministic, we will need to use non-deterministic timed automata for it.

7.3 From Future MITL to Timed Automata

We will now show how to build for every MITL formula φ a property tester, a timed automaton over the propositional variables and an auxiliary variable u which accepts all signals (ξ, u) satisfying $\Box(u \equiv \varphi)$.¹⁵ The proof is by induction on the structure of the formula, and the major step is the proof of the following proposition.

Proposition 4 *One can construct a timed automaton accepting signals over p, q, u that satisfy $\Box(u \equiv p\mathcal{U}_{[a,b]}q)$. Moreover, the automaton is deterministic if u is considered as input.*

¹⁵In other words, $u = \chi_\varphi(\xi)$.

The construction used to prove this proposition follows the lines of the untimed construction [KP05], based on generating predictions for u and aborting them when actual values of p and q show they were wrong. However, working on dense time we have the problem that a-priori, the set of potential predictions of bounded duration includes signals with an arbitrary number of switchings between true and false, and such predictions cannot be memorized by a finite-state timed device. The following lemma shows that predictions that switch too frequently cannot be true. A similar property was used in Section 7.2 to show that past MITL is deterministic.

Lemma 5 *Let u be a Boolean signal satisfying $\Box(u \equiv p\mathcal{U}_{[a,b]}q)$ for $a > 0$ and some arbitrary p and q . Then for any factorization $u = v \cdot 1^{r_1} \cdot 0^{r_2} \cdot 1^{r_3} \cdot 0^{r_4} \cdot w$ we have $r_2 + r_3 > \min\{a, b - a\}$.*

Proof: The following observations concerning the constraints on the values of u , p and q at every t follow from the definitions:

1. If u holds at t , p must hold in all the interval $[t, t + a]$;
2. If q holds at $t + b$ and p holds throughout $[t, t + b]$ then u holds during $[t, t + b - a]$.

Let $[t_1, t_2)$ and $[t_2, t_3)$ be the corresponding intervals for 0^{r_2} and 1^{r_3} , respectively (see Figure 11), and let us show that $(t_2 - t_1) < a$ implies that $(t_3, t_2) \geq b - a$. Since $(t_2 - t_1) < a$ and $u[t_1] = 1$, observation 1 implies that $p = 1$ throughout the interval $[t_1, t_2]$. As $u[t] = 1$ for all $t \in [t_2, t_3)$, it follows that $p[t] = 1$ for all $t \in [t_1, t_3)$. This implies that q must start holding at $t_2 + b$ and not before that, because otherwise this will imply that u holds inside the interior of $[t_1, t_2]$ contrary to our assumptions. It follows by observation 2 that u holds continuously in $[t_2, t_2 + b - a]$. Consequently, $t_3 \geq t_2 + b - a$ and we are done. \blacksquare

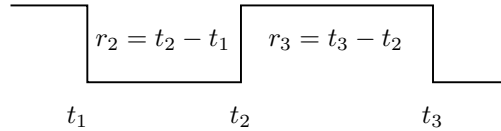


Figure 11: A signal u satisfying $\Box(u \equiv p\mathcal{U}_{[a,b]}q)$.

The importance of this property is that it bounds the variability of any reasonable prediction and constraints the relation between the number of changes and the duration of candidate signals. Let $d = \min\{a, b - a\}$ and $m = \lceil b/d \rceil + 1$. Since every¹⁶ 01 segment of u has at least d duration, an acceptable prediction of the form $(01)^m \cdot 0$ has a duration beyond b , its initial segment can be forgotten and its clock re-used as explained in the sequel.

We can now proceed to the proof of Proposition 4. Let us first describe the architecture of the automaton (Figure 12). The first component is a *prediction generator*

¹⁶To be more precise, the first 01 segment can be arbitrarily small.

for u , which generates signals non-deterministically, and uses clocks x_1, \dots, x_m and $y_1 \dots y_m$ where each clock x_i measures the time since the beginning of the i^{th} negative prediction segment and clock y_i measuring the time since the end of that segment. These predictions are passed through a “filter” which eliminates signals violating Lemma 1. In addition we use a battery of “testers” $\mathcal{A}_1^0, \dots, \mathcal{A}_m^0$ and $\mathcal{A}_1^1 \dots \mathcal{A}_m^1$ which observe p and q and check segments of the prediction. A tester \mathcal{A}_i^0 is active when $x_i \geq 0$ and $y_i < b$ and a tester \mathcal{A}_i^1 when $y_i \geq 0$ and $x_{i+1} < b$.

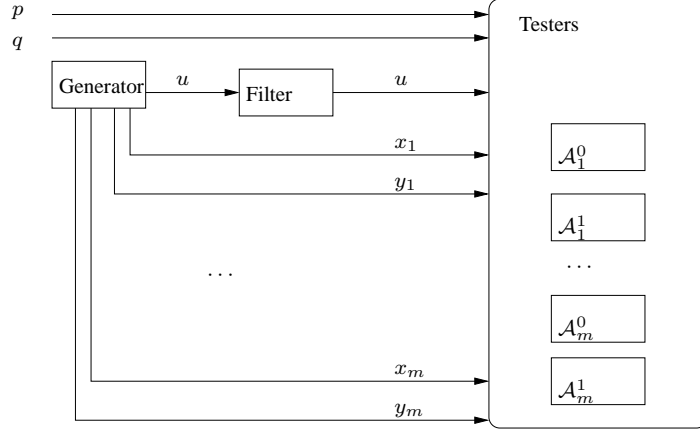


Figure 12: The architecture of the tester.

The generator is depicted in Figure 13. During the first m transitions the automaton has the following invariant property: whenever in a state of the form $(01)^k$ its prediction so far is of the form $0^{x_1-y_1} \cdot 1^{y_1-x_2} \cdot 0^{x_2-y_2} \dots 1^{y_k}$. After the m^{th} change, the first prediction segment is more than b -time old (Lemma 1) and cannot be refuted anymore by any observation of p or q . Hence there is no reason to memorize it and clock x_0 can be re-used for the $(m+1)^{\text{th}}$ false prediction segment and a transition back to the initial state can be made. This way the denotation of the clocks shifts circularly and they always remember the occurrence times of the last m changes in the prediction. The automaton of Figure 14 makes sure that only predictions that satisfy Lemma 1 are considered.

Automata \mathcal{A}_i^0 and \mathcal{A}_i^1 for testing prediction segments share a similar global structure. Both have an initial idle state that they leave when the prediction starts ($x_i = 0$ for \mathcal{A}_i^0 and $y_i = 0$ for \mathcal{A}_i^1) and to which they return when the prediction segment is too old to be refuted ($y_i = b$ for \mathcal{A}_i^0 and $x_{i+1} = b$ for \mathcal{A}_i^1). When in its active macro-state, automaton \mathcal{A}_i^0 (Figure 15) uses the auxiliary clock z to measure the duration of contiguous periods when p holds, and if q occurs when $z > a$, the false prediction is refuted and the run is aborted.¹⁷ The tester \mathcal{A}_i^1 for a positive prediction segment (Figure 16) aborts runs where p does not hold anywhere in the active interval. In addition, the auxiliary clock z measures periods of \bar{q} and if their duration reaches $b - a$ the run is aborted as well. It is not hard to see

¹⁷Note that all clocks in the generator, except x_0 are initialized to \perp to avoid satisfaction of the guards $x_i = 0$ or $y_i = 0$ during the first cycle of the run of the generator.

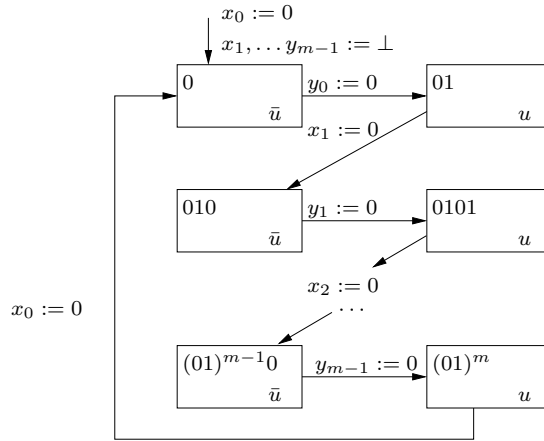


Figure 13: The prediction generator.

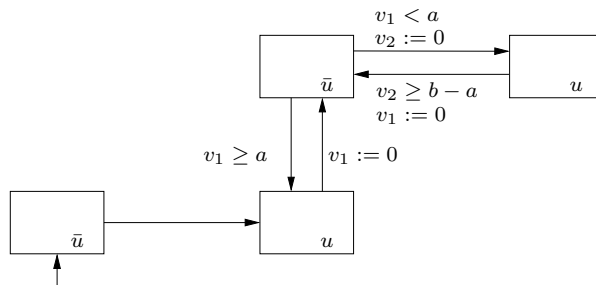


Figure 14: A “filter” for eliminating predictions violating Lemma 1.

that by composing these automata and letting F consist of all their states¹⁸ yields an automaton that accepts exactly models of the formula $\Box(u \equiv p\mathcal{U}_{[a,b]}q)$.

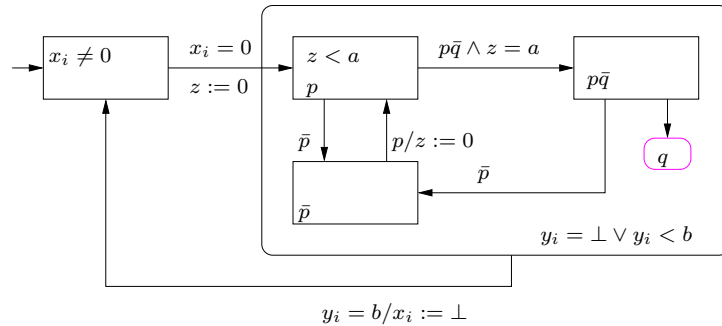


Figure 15: A negative tester \mathcal{A}_i^0 .

To complete the construction for MITL we need to build testers for Boolean combinations (easy, negations are pushed inside the formula), for the untimed until (as in the untimed case) and for the special case $\mathcal{U}_{[0,b]}$ which is simpler and is covered by the automaton of Figure 17.

¹⁸For this modality we do not need ω -conditions because the operator makes only bounded-horizon predictions.

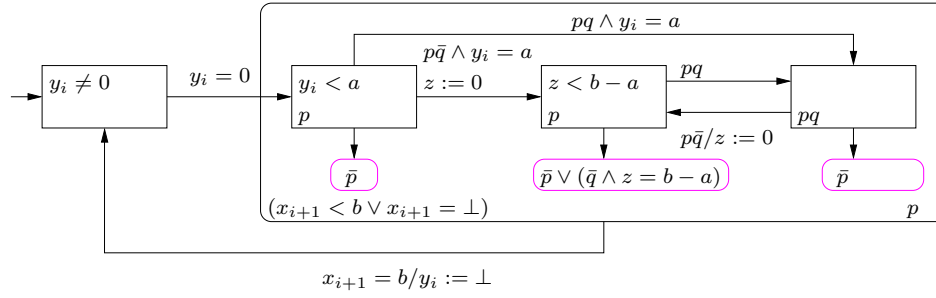


Figure 16: A positive tester \mathcal{A}_i^1 .

Corollary 6 (MITL and Timed Automata) *Future MITL formulae can be transformed into timed automata using a simple procedure.*

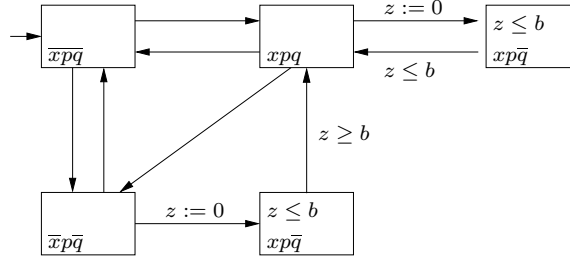


Figure 17: A tester for $\square(u \equiv p\mathcal{A}_{[0,b]}q)$.

While these construction can be very useful for property checking in the context of static verification, its application to monitoring requires further work. The major problem is that timed automata, in general, cannot be determinized. However the solution developed at the beginning of the project [KT04], based on the observation that timed automata can still be determinized “on-the-fly” with respect to a given timed behavior. The on-the-fly determinization algorithm becomes essentially a reachability computation on the property automaton and is based on state estimation techniques. After each partial input, the set of all possible states that can be reached by the observed trace is computed. The finite input sequence is accepted if and only if the computed set of states reached with respect to that trace contains at least one final state.

More technical details concerning the techniques described in this section can be found in the relevant papers.

8 Analog Signals

The algorithms developed for dense-time Boolean signals, provide a solid basis for monitoring analog signals when the properties belong to the class described in Deliverable D1.3/1 [M05] and Section YY, namely the *signal temporal logic* (STL) which is MITL parameterized by a set of numerical predicates in the role of propositional variables.. For such properties, each analog signal is transformed, via the numerical predicates appearing in 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 which transform analog signals to signals that can be subject to MITL checking.

As we have seen, 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 analog 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 consist 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 in 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 equation¹⁹, 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.²⁰

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

¹⁹It is worth noting that some models used for quick simulation of transistor networks cannot be viewed as continuous dynamical systems in the classical mathematical sense.

²⁰For systems which are stable the quality can be improved indefinitely.

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 $\square(x > 0)$ 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 Figure 18).

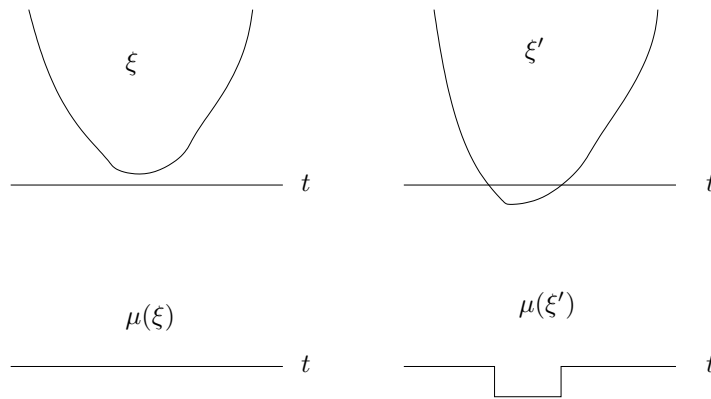


Figure 18: Two signals which are close from a continuous point of view, one satisfying the property $\square(x > 0)$ and one violating it.

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 Figure 19). In this sense properties are not *robust* as small variations²¹ in the signals may lead to large variations in its property satisfaction. Much of our effort in the project consisted in studying this problem and attempting to define new metrics, more appropriate for this hybrid discrete/continuous setting. Some very interesting research directions, published in [KC06], are briefly discussed in Section 10.2.

The abovementioned issues can be handled in a pragmatic manner in our context, without waiting for a completely-satisfactory theoretical solution to this fundamental problem. The following assumptions facilitate the monitoring of sampled analog 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

²¹Small from the continuous point of view.

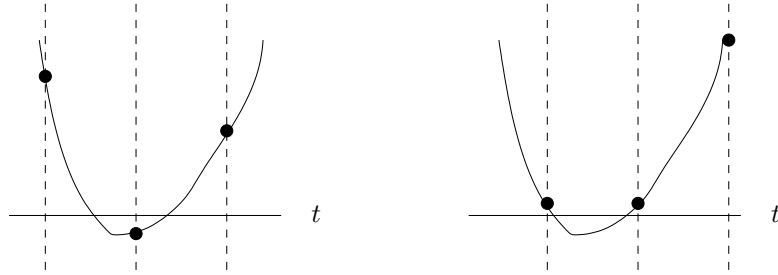


Figure 19: Shifting the sampling points, zero crossing can be missed.

level of sophistication on the simulator that has to perform several back-and-forth iterations to locate the time instance where a threshold crossing occurs. A survey of the treatment of discontinuous phenomena by numerical simulators can be found in [Mos99].

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 all the property checking based on these values is useless. Such conditions are not so important for Booleanized signals if sampling is sufficiently dense, but may become important when we consider richer classes of properties.

Under such assumptions every analog 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 digital verification cannot and should not be exported to the analog domain, and even if we are not guaranteed that all events are detected, we can compensate for that by using safety margins in the properties.

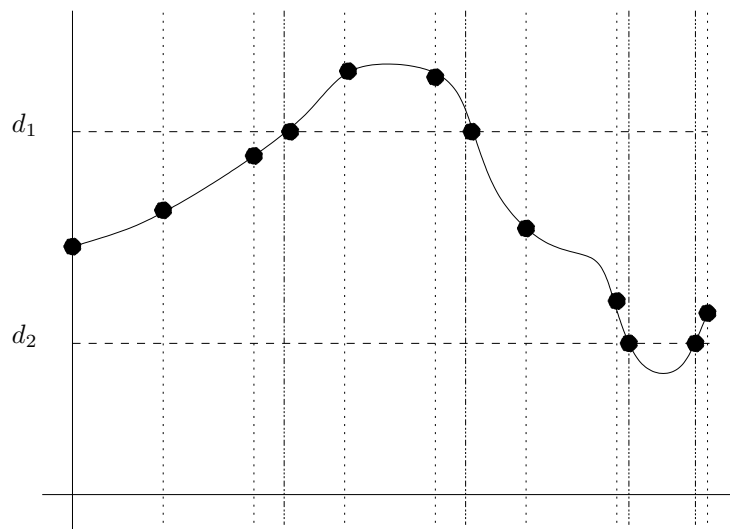


Figure 20: Sufficiently-dense sampling with respect to the two thresholds d_1 and d_2 . The set of sampling points consists of a uniform grid augmented with the threshold-crossing points.

9 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 prototype implementation which parses STL properties and applies the offline marking procedure described in Section 6.1. We use the same examples that were used in Deliverable D1.3/1 [M05] for illustrating the language extensions.

9.1 Following a Reference Signal

As a first example we consider the property

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

which requires that whenever x_1 crosses the threshold of 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 Figure 21. The Boolean signals corresponding to the atomic propositions p_1 and p_2 are derived from the sampled 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 Figure 22 we apply the same procedure to check φ_1 against an x_2 signal 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.

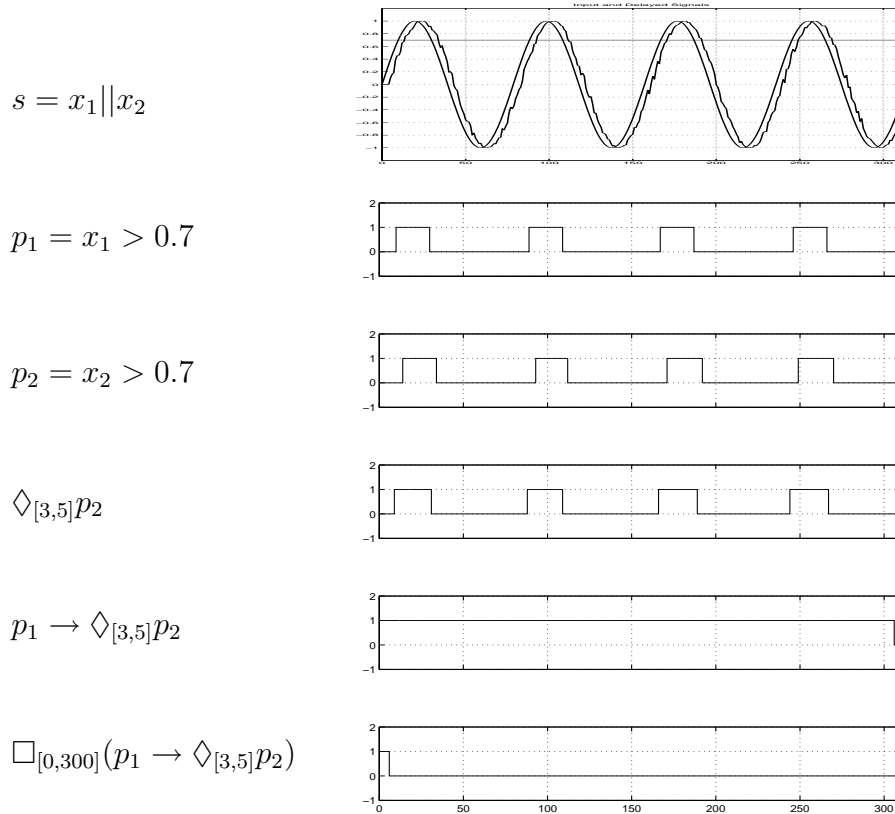


Figure 21: A 2-dimensional signal satisfying the property $\square_{[0,300]} ((x_1 > 0.7) \Rightarrow \diamond_{[3,5]} (x_2 > 0.7))$. Boolean signals correspond to the evolution of the truth values of sub-formulae over time.

9.2 Stabilizability

The second example is a very typical stabilizability property used extensively in control and signal processing. The system in question is supposed maintain a controlled variable y around a fixed level despite disturbances x coming from the outside world. One may think, for example, of a system that has to maintain the voltage constant albeit variations in the current due to changes in the chip's

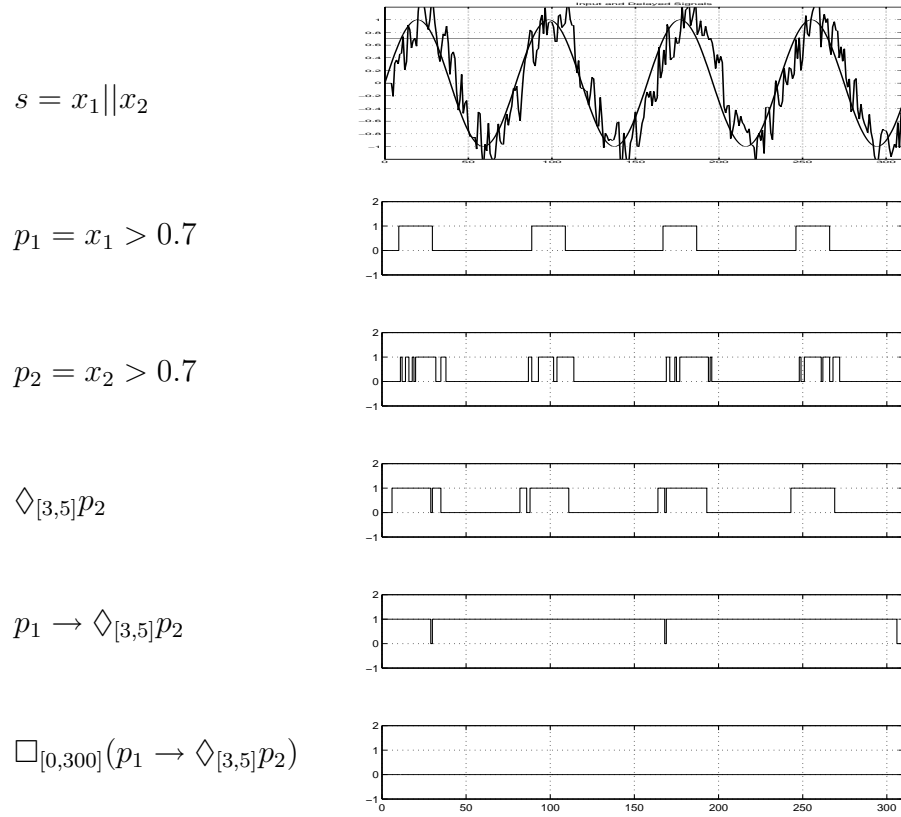


Figure 22: A 2-dimensional signal violating the property $\square_{[0,300]}((x_1 > 0.7) \Rightarrow \diamond_{[3,5]}(x_2 > 0.7))$.

workload.²² The role of the control system is to stabilize the controlled variable to the desired level after every disturbance.²³

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

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

The results of applying our offline monitoring procedure to this formula appear in Figures 23 and 24. When the disturbance is well-behaving, the property is verified,

²²Modern power-management systems for cellular phones react this way to changes coming from starting and stopping certain features.

²³The actual system used to generate this example is a water-level controller for a nuclear plant. The disturbances come from changes in the system load that trigger changes in the operations of the reactor which, in turn, influences the water level.

while when the disturbance changes too fast, the property is violated both by overshooting below -30 and by taking more than 150 time to return to $[-0.5, 0.5]$.

To demonstrate the complexity of our procedure as a function of signal length we applied it to increasingly longer signals ranging from 5000 to one million seconds. We use variable integration/sampling step with average step size of 2 seconds so the number of sampling point in the input is roughly half the number of seconds. The results are depicted in Table 2 and one can see that monitoring can be done very quickly and it adds a negligible overhead to the simulation of complex systems. For example, the simulation of the stabilizing controller (based on a complex model of a nuclear plant) for a time horizon of million seconds takes 45 minutes while monitoring the output takes less than 3 seconds. Of course, using the incremental procedure we can shorten the checking time further and, in an online setting, this procedure can reduce simulation time.

sig length	$ \mathcal{I}_p^+ $	$ \mathcal{I}_q^+ $	time(sec)
5000	98	82	0.01
50000	970	802	0.13
100000	1920	1602	0.25
200000	3872	3202	0.49
500000	9732	8002	1.36
1000000	19410	16002	2.84

Table 2: CPU time of monitoring the water level controller example as a function of the time horizon (signal length). The number of positive intervals in the Boolean abstractions is given as another indication for the complexity of the problem.

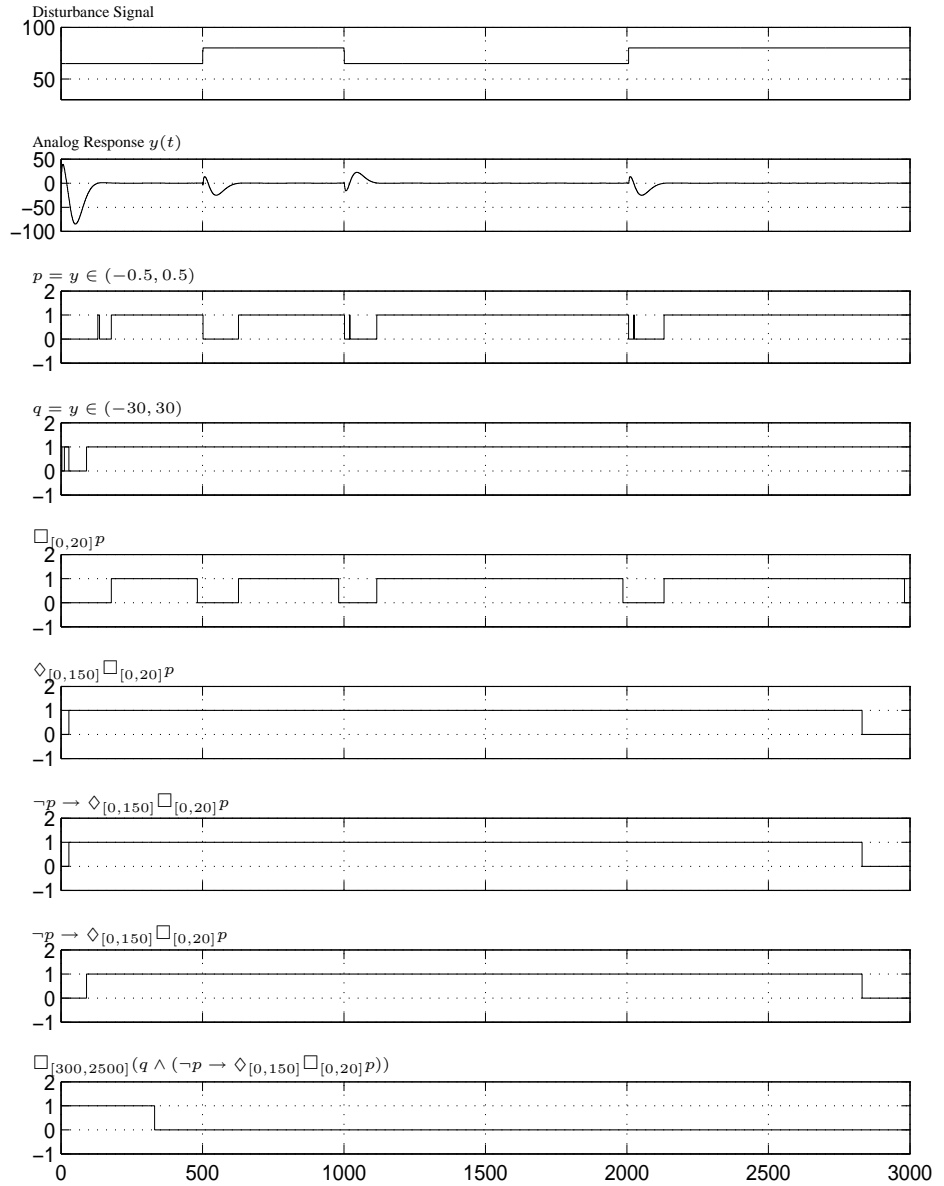


Figure 23: A disturbance signal and an analog response y satisfying the stabilizability property $\square_{[300,2500]} ((|y| \leq 30) \wedge (|y| > 0.5) \Rightarrow \diamond_{[0,150]} \square_{[0,20]} (|y| \leq 0.5))$.

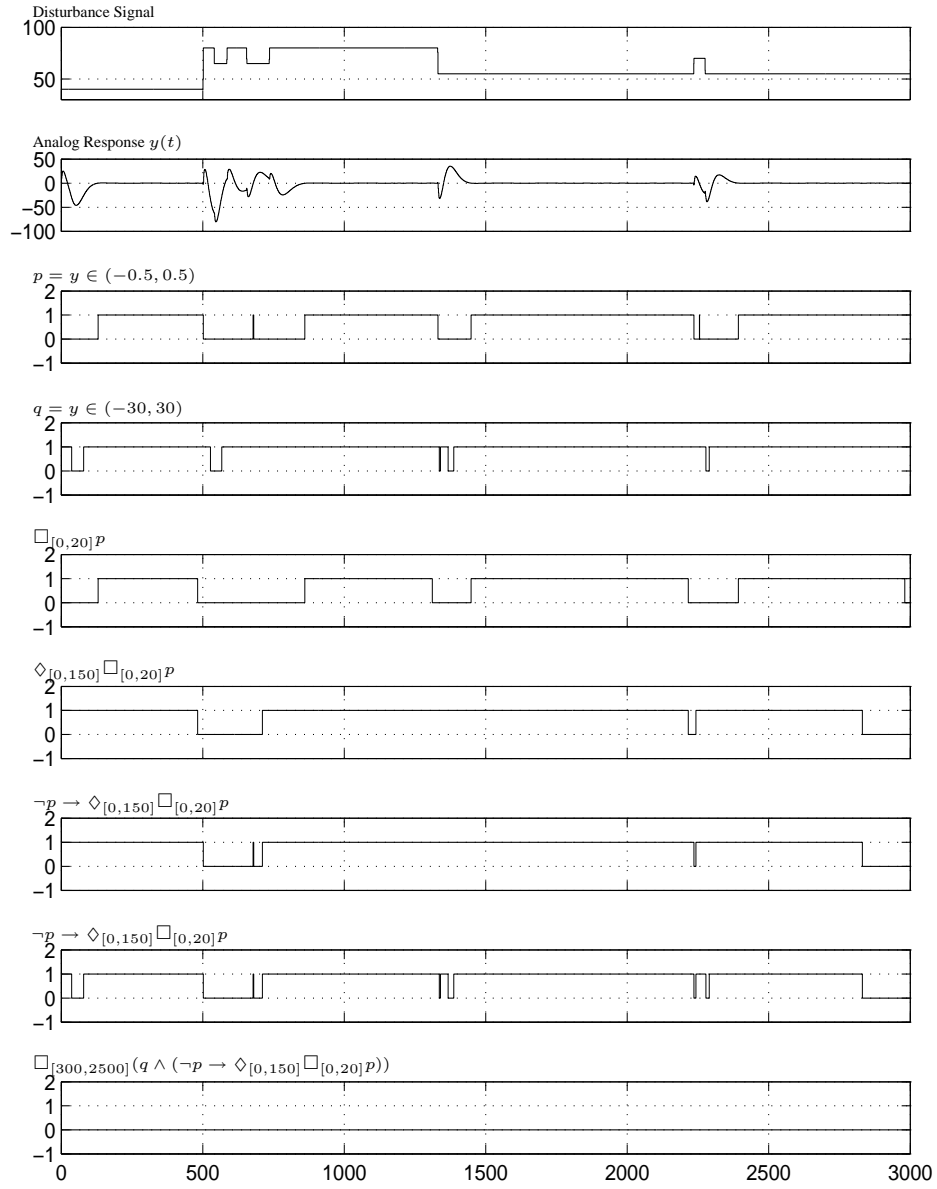


Figure 24: A disturbance signal and an analog response y violating the stabilizability property $\square_{[300,2500]}((|y| \leq 30) \wedge ((|y| > 0.5) \Rightarrow \diamond_{[0,150]} \square_{[0,20]}(|y| \leq 0.5)))$.

10 Further Directions

This section summarizes some additional developments at more preliminary levels of maturity.

10.1 The FLASH-Memory Case Study

The role of the analog part of PROSYD is to build a bridge between two almost-disjoint worlds, the world of formal verification and the world of analog circuit design. Most of the effort has been invested in one direction, that is, taking existing results in the domain of formal methods and adapting them to the richer context of analog circuits and signals. Hopefully this work will allow designers to express and check “sequential” properties that they cannot express using their current vocabulary. The complementary direction is no less important: it consists in studying current practices in analog design²⁴ and suggest new ways to support these activities, not necessarily using currently-existing techniques coming from formal verification. This “bottom-up” activity depends critically on the willingness of analog designers to allocate parts of their limited time to explain their problems to sympathetic strangers.

We started exploring a case-study provided by ST Agrate concerning the desired properties of a FLASH memory. The discussion with designers confirmed our a-priori belief that monitoring simulations is a very time-consuming and error-prone activity and that a monitoring tool like the one we are constructing can indeed be helpful. We have formalized together a set of properties, most of which turned out to be expressible in our STL logic.²⁵

Among the properties not covered by our logic are properties that speak about the *slopes* (derivatives) of certain signals. A general approach to accommodate for such operators (and other signal operators) is to pre-process the signal before Booleanization and generate an auxiliary signal for each operator mentioned in the

²⁴The term “analog” is used here in the more general sense that includes also the analysis of digital systems at the electrical level.

²⁵To be fair, we must mention that we are concerned here with the electrical properties of a *digital* device whose high-level properties are sequential in nature, for example, “the *read* signal goes high and stays there until the *ack* signal arrives”. Hence this favorable situation need not repeat itself for other classes of circuits.

formula. However applying this idea to the derivative operator is not so simple due to the noise in the simulator output. For example if we define the derivative signal ξ' using the straightforward approximation

$$\xi'[t_i] = \frac{\xi[t_i] - \xi[t_{i-1}]}{t_i - t_{i-1}}$$

we obtain, for this case-study, a signal with abrupt fluctuations. We are currently studying the applicability of *filtering* techniques in order to smoothen things out.

10.2 Defining and Checking Metrical Properties

Discussions with designers confirmed our feeling that a large part of the evaluation of the quality of signals produced by the system under design is based on comparing the signal with a *reference signal* indicating some kind of desired normative behavior. However, similarity, like beauty, is in the eye of the beholder. Technically speaking, it depends on the metric used to determine how close one signal is to another. We believe that in the future, any useful property checker for analog properties should have the ability to define such reference signals and compute the distance between it and simulated signals according to the metrics that are appropriate for the problem. For metrics that are “pointwise”, that is compare ξ and ξ' according to $\xi[t] - \xi'[t]$ for every t , computing an approximation of the distance is not much different from computing any other operation. On the other hand if we are talking about a realization of a digital behavior, the similitude between signal might be more related to the fact that they go through the same sequence of events (threshold crossing).

To prepare for such future developments we started investigating new metrics and topologies which are appropriate for hybrid (mixed-signal) systems, as well as for the interaction between analog signals and discrete properties. In particular, what metric should be used for piecewise-continuous and piecewise-constant signals?²⁶ One idea is to base the metric on the variation of the discontinuity instants: two signals are close if they go through the same set of abstract values in the same order and their switching times are also close. We have investigated the applicability of the following metrics:

1. The Skorokhod metric which comes from probability theory.
2. The Hausdorff metric between trajectories, proposed in [GHJ97] for characterizing robust hybrid systems.

²⁶Boolean and any discrete-valued signals are, of course, piecewise constant. This is also a common interpretation of sampled signals.

3. A sliding window mean value metric [KC06].

The latter seems to give good results in the sense that it is less strict than the two others and tolerates more variations which naturally appear in complex systems. The computational aspects of such metrics still need to be investigated.

11 References

- [ABG⁺00] Y. Abarbanel, I. Beer, L. Glushovsky, S. Keidar, and Y. Wolfsthal, FoCs: Automatic Generation of Simulation Checkers from Formal Specifications, *CAV'00*, 538-542, LNCS 1855, Springer, 2000.
- [AD94] R. Alur and D.L. Dill, A Theory of Timed Automata, *Theoretical Computer Science* **126**, 183-235, 1994.
- [AFH96] R. Alur, T. Feder, and T.A. Henzinger, The Benefits of Relaxing Punctuality, *Journal of the ACM* **43**, 116-146, 1996.
- [AH92] R. Alur and T.A. Henzinger, Logics and Models of Real-Time: A Survey, *REX Workshop, Real-time: Theory in Practice*, 74-106. LNCS 600, Springer, 1992.
- [A04] E. Asarin, Challenges in Timed Languages, *Bulletin of EATCS* **83**, 2004.
- [ACM02] E. Asarin, P. Caspi and O. Maler, Timed Regular Expressions, *The Journal of the ACM* **49**, 172-206, 2002.
- [BBDE⁺02] I. Beer, S. Ben David, C. Eisner, D. Fisman, A. Gringauze, and Y. Rodeh, The Temporal Logic Sugar, *CAV'01*, LNCS 2102, Springer, 2002.
- [BBF⁺01] B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen, and P. McKenzie, *Systems and Software Verification: Model-Checking Techniques and Tools*, Springer, 2001.
- [BBKT04] S. Bensalem, M. Bozga, M. Krichen, and S. Tripakis, Testing Conformance of Real-time Applications with Automatic Generation of Observers, *RV'04*, 2004.
- [CGP99] E.M. Clarke, O. Grumberg, and D.A. Peled, *Model Checking*, The MIT Press, 1999.
- [DY96] C. Daws and S. Yovine, Reducing the Number of Clock Variables of Timed Automata, *RTSS'96*, 73-81, IEEE, 1996.
- [Don03] A. Donzé. Etude d'un Modèle de Contrôleur Hybride. Master's thesis, INPG, 2003.
- [Dru00] D. Drusinsky, The Temporal Rover and the ATG Rover, *SPIN'00*, 323-330. LNCS 1885, Springer, 2000.
- [EFH⁺03] C. Eisner, D. Fisman, J. Havlicek, Y. Lustig, A. McIsaac, and D. Van Campenhout, Reasoning with Temporal Logic on Truncated Paths, *CAV'03*, 27-39, LNCS 2725, Springer, 2003.
- [GD00] M.C.W. Geilen and D.R. Dams, An On-the-fly Tableau Construction for a Real-time Temporal Logic, *FTRTFT'00*, 276-290. LNCS 1926, Springer, 2000.

- [Gei02] M.C.W. Geilen, *Formal Techniques for Verification of Complex Real-time Systems*, PhD thesis, Eindhoven University of Technology, 2002.
- [Gei03] M.C.W. Geilen, An Improved On-the-fly Tableau Construction for a Real-time Temporal Logic, *CAV'03*, 394-406, LNCS 2725, Springer, 2003.
- [GHJ97] V. Gupta, T.A. Henzinger and R. Jagadeesan, Robust Timed Automata, *HART'97*, 331-345, LNCS 1201, Springer, 1997.
- [HR01] K. Havelund and G. Rosu, Java PathExplorer - a Runtime Verification Tool, *ISAIRAS'01*, 2001.
- [HR02a] K. Havelund and G. Rosu (editors), *Runtime Verification RV'02*, ENTCS 70(4), 2002.
- [HR02b] K. Havelund and G. Rosu, Synthesizing Monitors for Safety Properties, *TACAS'02*, 342-356, LNCS 2280, Springer, 2002.
- [Hen98] T.A. Henzinger, It's about Time: Real-time Logics Reviewed, *CONCUR'98*, 439-454, LNCS 1466, Springer, 1998.
- [HNSY94] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, Symbolic Model-checking for Real-time Systems, *Information and Computation* **111**, 193-244, 1994.
- [HR04] Y. Hirshfeld and A. Rabinovich Logics for Real Time: Decidability and Complexity, *Fundamenta Informaticae* **62**, 1-28, 2004.
- [KP05] Y. Kesten and A. Pnueli, A Compositional Approach to CTL* Verification, *Theoretical Computer Science* **331**, 397-428, 2005.
- [KC06] C. Kossentini and P. Caspi, Approximation, Sampling and Voting in Hybrid Computing Systems, *HSCC*, to appear, 2006.
- [Koy90] R. Koymans, Specifying Real-time Properties with with Metric Temporal Logic, *Real-time Systems*, 255-299, 1990.
- [KLS⁺02] M. Kim, I. Lee, U. Sammapun, J. Shin, and O. Sokolsky, Monitoring, Checking, and Steering of Real-time Systems, *RV'02*, ENTCS 70(4), 2002.
- [KPA03] K.J. Kristoffersen, C. Pedersen, and H.R. Andersen, Runtime Verification of Timed LTL using Disjunctive Normalized Equation Systems, *RV'03*, ENTCS 89(2), 2003.
- [KT04] M. Krichen and S. Tripakis, Black-box Conformance Testing for Real-time Systems, *SPIN'04*, 109-126, LNCS 2989, 2004.
- [KV01] O. Kupferman and M.Y. Vardi, On Bounded Specifications, *LPAR'01*, 24-38, LNCS 2250, 2001.
- [Kur94] R. Kurshan, *Computer-aided Verification of Coordinating Processes: The Automata-theoretic Approach*, Princeton University Press, 1994.
- [Mos99] P.J. Mosterman, An Overview of Hybrid Simulation Phenomena and their Support by Simulation Packages, *HSCC'99*, 165-177, LNCS 1569, 1999.

- [LPZ85] O. Lichtenstein, A. Pnueli and L.D. Zuck, The Glory of the Past, *Conf. on Logic of Programs*, 196-218, LNCS, 1985.
- [M05] O. Maler, *Extending PSL for Analog Circuits*, PROSYD Deliverable D1.3/1, 2005.
- [MN04] O. Maler and D. Nickovic, Monitoring Temporal Properties of Continuous Signals, *FORMATS/FTRTFT'04*, 152-166, LNCS 3253, 2004.
- [MNP05] O. Maler, D. Nickovic and A. Pnueli, Real Time Temporal Logic: Past, Present, Future, *FORMATS'05*, 2-16, LNCS 3829, 2005.
- [MNP06] O. Maler, D. Nickovic and A. Pnueli, A Simple Decision Procedure for MITL, submitted for publication, 2006.
- [MP04] O. Maler and A. Pnueli, On Recognizable Timed Languages, *FOS-SACS'04*, 348-362, LNCS 2987, Springer, 2004.
- [MP92] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems - Specification*, Springer, 1992.
- [MP95] Z. Manna and A. Pnueli, *Temporal Verification of Reactive Systems: Safety*, Springer, 1995.
- [Pnu03] A. Pnueli. Verification of Reactive Systems. Lecture Notes, NYU, 2003. <http://cs.nyu.edu/courses/fall03/G22.3033-007/lecture4.pdf>.
- [S88] S. Safra, On the Complexity of ω -Automata, *FOCS'88*, 319-327, 1988.
- [SV03] O. Sokolsky and M. Viswanathan, editors. *Runtime Verification RV'03*. ENTCS 89(2), 2003.
- [TR04] P. Thati and G. Rosu, Monitoring Algorithms for Metric Temporal Logic Specifications, *of RV'04*, 2004.
- [Tri02] S. Tripakis, Fault Diagnosis for Timed Automata, *FTRTFT'02*, 205-224, LNCS 2469, Springer, 2002.
- [VW86] M.Y. Vardi and P. Wolper, An Automata-theoretic Approach to Automatic Program Verification, *LICS'86*, 322-331, IEEE, 1986.