

PROSYD Deliverable D1.3/1
Extending PSL for Analog Circuits

Oded Maler
No Logo

January 12, 2005

1 Introduction

This document is a first proposal for an extension of PSL toward real-time and analog properties. It represents mainly the work of Verimag and Weizmann with a participation of ST in giving insights about the the design and validation of such circuits. Additional useful information about analog design was provided by sources outside PROSYD, including Dolphin SA, CEA/LETI, Intel, Professors Rutenbar and Krogh from CMU and Prof. Hedrich from Hannover.

The rest of the document is structured as follows:

- Section 2: scientific and technological background of this work, emphasizing the sharp contrast in maturity levels between digital and analog verification.
- Section 3: continuous-valued signals and the problems related to their processing by computers.
- Section 4: properties in the context of such signals and characterization the type of properties that we focus on.
- Section 5: the major part of this document where we introduce the specification language STL (Signal Temporal Logic) that we develop. We start with an intuitive explanation of the way it treats the particular features of signals, namely infinite state-space and dense metric time and then proceed with formal definitions of the syntax and semantics of STL.
- Section 6: some examples of STL formulae and of signals that satisfy and violate them.
- Section 7: a sketch of some further possible extensions of the language.
- Section 8: some additional related work that has been conducted within PROSYD but is not part of the current deliverable, namely the development of property monitors for STL and the exhaustive verification of some analog circuits test cases.
- Section 9: Conclusions.

2 Background

We start by situating the role of this work-package within PROSYD. Property based system design means that a system is characterized by a set of properties it should satisfy. These properties specify, in a formal language, which traces of I/O behaviors the system may exhibit while interacting with its external environment. There are basically two approaches for validating a system with respect to a given property. Both of them are based on transforming the property into a *property monitor*, a mechanism that checks whether a given behavior (sequence of I/O events) satisfies the property. This monitor can be viewed either as an automaton accepting exactly the set of satisfying behaviors [VW86] or as a procedure working recursively both on the length of the sequence and on the syntactic structure of the property.

In *simulation/testing* a model of the system is used to generate simulation traces, and each of those is checked by the monitor. If one of the traces violates the property the system is incorrect. However since the system is to be exposed to a potentially infinite (or prohibitively large) number of inputs, it is impossible to complete this procedure for all possible inputs. A large effort in this domain is about the systematic generation of test-cases that somehow “cover” all classes of behaviors.

The goal of *algorithmic verification* is more ambitious. The transition graph of the system is explored in order to show that *all* the sequences it can generate are accepted by the monitor. A major problem here is that of state-explosion as the number of states of the system is exponential in the number of state-variables (memory holding elements, in the case of digital circuits). Much of the current research in verification is about finding clever ways to cope with this situation and about developing the methodological aspects of property-based system design, as the PROSYD project attempts to do. This work-package is concerned with extending this methodology to analog and mixed-signal systems. To assess the contrast between the respective situations in the digital and analog domains it is worth recalling the evolution of (digital) formal verification up to the present:

- Early work on program verification (1965-1977).
- Introduction of Temporal Logic as a formalism for specifying properties of reactive systems (Pnueli 1977).
- Work on deductive (partly-manual) verification for TL (1977-present).
- First model-checking algorithms for fully automatic verification (Queille and Sifakis 1980, Emerson and Clarke 1981).
- First workshop on computer-aided verification (Grenoble, 1989).
- First symbolic model-checker that could treat systems with state-space too large to be enumerated (McMillan 1992).
- The development of industrial-strength verification tools (1993-present)
- The Intel bug and the proliferation of formal verification into the semi-conductor industry (1995).

- The development of Sugar (1994-1998).
- Accelerated discussions that culminate in PSL (1998-2004).
- The PROSYD project starts (2004)

As we can see, it took almost 30 years to push theoretical ideas into industrial-strength tools and along the way, in addition to impressive algorithmic development, cultural gaps between theoreticians and practitioners had to be bridged. Verification is founded upon logic, automata and semantics which are part of theoretical computer science. To a certain extent some knowledge of these topics is part of the background of digital designers via Boolean logic and sequential finite-state machines. In the other direction, most theoretical computer scientists have a basic understanding of digital circuits, at least at the gate and flip-flop level of abstraction. Conferences like *CAV: Computer-Aided Verification*, as well as collaborations between researchers and industry, contributed a lot to the creation of a common verification culture.

The situation in the analog domain is radically different. The electrical behavior of transistors in digital circuits is just a means to realize logical gates. This implies that the functional correctness of a circuit can be validated using an abstract model that corresponds to a sequential machine, without worrying too much about the physical realization. Such physical details may influence “non-functional” properties of the circuit such as timing or power consumption, but the logical abstraction is robust with respect to such variations. In contrast, the functionality of analog circuits is defined directly in terms of continuous electrical quantities. As such they are much more sensitive to unavoidable (and sometimes random) perturbations from the overall operating environment such as voltage from the supply, temperature, noise from nearby electronics or noise from fundamental sources such as the physics of the insides of the basic transistors. One unpleasant consequence of this situation is that important parameters of the circuit dynamics are determined only after physical placement or even fabrication.

The behavior of analog circuits is simulated and analyzed using models of continuous dynamical systems specified by differential and algebraic equations. Systematic mathematical support for these activities exists typically only for linear systems. The culture of designers may include many non-digital parts of electrical engineering, including signal processing and its mathematical basis (Fourier analysis, information theory), linear algebra and linear systems theory, numerical computations and of course, intimate knowledge of the behavior of real transistors and of the physics of the particular application domains in which the circuits are used. Most of these domains are outside the background of computer scientists. The history of the attempts to export formal verification methodology toward models admitting real-valued variables and dense time is much shorter and is outlined below:

- First attempts to define specification formalisms and computational models for real-time systems (1985-1995).
- Positive results concerning the verification of timed automata (Alur and Dill 1990 [AD94]).
- A first proposal to verify hybrid systems (Maler, Manna and Pnueli 1991).

- Development of the algorithmic approach for the analysis of timed and “linear” hybrid automata (Henzinger et al); First tools: Kronos (timed automata, Verimag), Hytech (hybrid automata, Berkeley); Negative undecidability results that show that exact verification is impossible even for systems with few continuous variable and very simple dynamics (1992-1998).
- Development of the first tools for approximate verification of continuous and hybrid systems having non-trivial continuous dynamics with few state variables (Checkmate, CMU 1999; d/dt, Verimag 2000; Hysdel, ETH 2000); Control systems analysis is the main application domain.
- First attempts to apply formal verification to analog circuits (2002-present).

As one can see, the verification of systems having continuous variables is, provably, much more difficult than that of finite-state systems, and that verification tools for such systems are still in their infancy. Before trying to export verification methodology to analog circuits we should first clarify the motivation for doing so. Chips with analog or RF function on them are notoriously difficult to get right on the first try, so anything we do that helps uncover more functional flaws or “bad behaviors” in subtle corners of the performance space is regarded as worthwhile. Designers of such circuits do use mathematical models for analytical and numerical computations, but the treatment of these models is more in the engineering and applied mathematics tradition, without the careful semantic and methodological concepts developed for modeling digital concurrent systems. Since the importance of analog components grows as systems become more integrated on a chip and more embedded in the physical world, any contribution toward accelerating their development will have significant economic consequences. It is believed that formal methods will occupy some complementary niche among the currently-used design methods for analog systems, as they already do for digital systems.

The goal of the analog part of PROSYD is to lay the foundation for such future progress by developing verification methods for analog and mixed-signal circuits and by proposing extensions to the property specification language (PSL) that will allow designers to specify desired properties of such circuits and to check whether given behaviors of the system satisfy them.

3 Behaviors

A behavior of a system is a function from a time domain to its state-space. In discrete systems, such as automata, the time domain is typically a discrete linearly-ordered set which is order-isomorphic to the natural numbers (we ignore here partial orders used sometimes for distributed systems). This time domain is more of a “logical” nature rather than metric. This means that we are interested more in the order among the events rather than the real-time temporal distance between them. In other words, if we take a behavior of a system and “accelerate” parts of it without changing the order of events, the behavior will be considered as practically equivalent and will satisfy the same set of properties. In digital synchronous circuits, this time domain often corresponds to the ticks of the main clock, and hence some metric aspects of time are represented. The state-space of a digital system consists of all possible combinations of values of memory holding elements. At the gate level this means essentially Boolean vectors and, more generally, the possible values of registers, encoded as well in binary. Hence a behavior of a digital system can be viewed as a sequence of states, $s : \mathbb{N} \rightarrow \mathbb{B}^n$.

In analog systems the state variables hold continuous quantities such as voltage and current which are perceived as real numbers. Unlike digital systems where such values are observed only at certain time instants, in analog systems we are often interested in the evolution of these quantities over the entire time axis. Mathematically speaking, such behaviors are viewed as signals of the form $s : \mathbb{R}_+ \rightarrow \mathbb{R}^n$. This much richer type of objects raises some conceptual problems that do not exist in digital sequences.

The first problem is related to the effective representation of signals, a pre-condition for their treatment by computers. Digital sequences can be simply represented by a list $s[0], s[1], \dots$ that specifies their value at each time instant. Analog signals cannot be wholly specified in this way due to the density of real numbers. In some cases they can be specified by a closed form expression, for example $s[t] = \sin(t)$, while for the purpose of simulation they are represented via discretization of both the time domain and the state-space. For the latter, one typically uses the floating-point numbers, a finite but rather dense subset of the rationals. Thus instead of saying that $s[t] = x$ for some $x \in \mathbb{R}^n$, we can only say that $s[t] = x'$ where x' is a floating point number in the neighborhood of x . Much of the research in numerical analysis and simulation tools is concerned with the effect of such imprecision, accumulation of errors, etc. We do not consider this to be a major issue in the definition of temporal properties of signals.

The finite representation of the time domain and the values of the signal on it is much more problematic. There are two basic (non-analytic) ways to give a finite representation of a signal. The first approach is based on uniform sampling. A discretization step δ is chosen, and the discretized time domain is set to be the sequence $\delta, 2\delta, \dots$. A signal is then specified as $s' = s(\delta), s(2\delta) \dots$. This leads to an inherent loss of information as we do not know the value of the signal between two sampling points. This means that s' represents an infinitude of signals, all those that agree with s' on the sampling points. This representation can be interpreted as defining a specific default signal (piecewise-constant, piecewise-linear or other interpolation) which can be made as close as we want to s (using various metrics) by making δ smaller.

The complementary approach for representing signals is by using a non-uniform subset of the time domain. For example, one may use large discretization steps when

the signal value is in “less important” parts of the state-space, and then use more dense sampling when the values are in the important range. In the same spirit, we can require that the discretized time domain will include all instants when important events do happen, for example when the value of a variable crosses some pre-defined threshold. It is clear that using this approach different signals will have different discretized time domains as they cross thresholds at different times. These approaches are sometimes referred to as “time-triggered” and “event-triggered” sampling.

These considerations will have practical consequences as we define what it means for a *given* signal to satisfy a property. However, since these features of the signal presentation may vary depending on the simulation method used and other details, we adopt the following strategy: we define the semantics of the specification language in terms of the ideal mathematical objects, that is, signals of the form $s : \mathbb{R}_+ \rightarrow \mathbb{R}^n$, and only when considering concrete algorithms for checking satisfaction we refer to the actual finite presentation of the signal.

It is also worth mentioning that the richness of the mathematical reals allows one to define pathological signals such as those with infinite frequency¹ which should be excluded from the discussion. However, one should be careful because it is not always clear what is to be assumed and what is to be proved. For example, if we know the values of s at two sampling point t and t' , we may use some bound on the time derivative of s to deduce bounds on its value during the whole interval $[t, t']$, but sometimes the bound on the derivative can be exactly what we want to verify.

¹For example, signals which are 1 on rationals and 0 on irrationals.

4 Properties

In digital verification properties are used to distinguish between good and bad behaviors. The simplest and most frequently used properties specify the absence of certain “bad” states in the sequence in question. More sophisticated properties speak of the occurrence of certain sequential patterns in the behavior, for example, “*a* is followed by *b*”. The two most popular specification styles are based on *temporal logic* and on *regular expressions*, respectively. One difference between these formalisms is that traditionally temporal logic is more adequate for multi-dimensional signals while regular expressions, at least classically, were oriented toward monolithic alphabets. The main distinguishing feature between the two is in the operator used for sequencing. Temporal logic uses the *until* operator (or its past analogue, the *since* operator), while regular expressions use concatenation (concatenation was also introduced into temporal logic via the “chop” operator). Both formalisms are very natural in the context of digital sequential machines and are supported by the PSL language. On the other hand, they are quite different from the way continuous signals are evaluated by practitioners and theoreticians in domains that are relevant for analog systems.

Perhaps the most natural way to introduce properties to these domains is to start from the more general notion of *performance measure*. A performance measure is a function that maps a signal into one or more numbers that measure its quality, and can serve as a basis for preferring one signal over the other. Such measures can be based on integrating the value of the signal over time, on its being periodic, on the frequency spectrum of the signal or on its distance from some reference signal that describes the ideal behavior of the system. From this point of view, properties are just a special type of performance measure which map signals into a two-valued Boolean domain, that is, good and bad.

Since the goal of this part of PROSYD is to extend property-based system design from digital to analog systems, the first proposal for language extension is to adapt the “sequential” formalisms used in the former to continuous and mixed signals. Such formalisms are new to analog designers and the first proposed extension will provide them with a new language that can express phenomena they could not express so far, at least not in a systematic and explicit manner. In further stages of the project we will also attempt to include in our language additional features that formalize other performance measures currently used by designers. In the rest of the document we focus on *signal temporal logic* (STL), developed within this work-package, a natural first-order extension of propositional linear temporal logic which addresses the two conceptual difficulties mentioned above: dense metric time and infinite state-space.

5 Signal Temporal Logic

5.1 Intuition

We start with an informal description of the design decisions for the proposed language extension.

5.1.1 Infinite State-Space

The building blocks of temporal logic for digital systems are the atomic propositions, symbols that refer to instantaneous values of Boolean state variables. From these atomic propositions one constructs state formulae by means of Boolean operations. Each state formulae thus defines a subset of the state-space. At any time instant t one can check whether the Boolean vector $s[t]$ satisfies a state formula.

For real-valued variables the state-space is infinite (and even non-countable) and enumeration of its elements is impossible. Subsets of \mathbb{R}^n are defined using symbolic expressions (formulae, constraints, inequalities) that need to be satisfied by a state $x = (x_1, \dots, x_n)$ in order to belong to the subset. Such constraints can vary in complexity, starting with simple “rectangular” constraints of the form $x_i < c$, through more general linear inequalities of the form

$$a_1x_1 + a_2x_2 + \dots + a_nx_n < c$$

to more complex algebraic and trigonometric ones. Since the range of possibilities here is infinite, we will parameterize the definition of the logic by the primitive constraints used. That is, if $\mathcal{P} = \{\mu_1, \dots, \mu_n\}$ is a set of constraints, each of which defining a characteristic function of the form $\mu_i : \mathbb{R}^n \rightarrow \mathbb{B}$, then the set of temporal formulae that can be constructed from these building blocks will be called $\text{STL}(\mathcal{P})$.

5.1.2 Dense Time

As previously mentioned, the conceptual difficulties associated with moving from discrete to dense time are much more important. In fact, the need for dense “asynchronous” time formalisms is not restricted to analog circuits and can be very useful for the functional analysis of digital systems which are not clocked or for finer timing analysis of digital circuits in general. If we look at the “ a followed by b ” specification, we see that it says nothing about the time between these two events. If we model digital circuits in more detail at the level of gates plus delays, their performance and even their correctness may depend critically on the timing parameters. During the last 15 years a lot of effort has been invested in extending specification and verification methodology from purely discrete systems to such “timed” systems. These extensions include the development of the timed automaton model, which is an automaton augmented with fictitious clock variables, and whose behaviors consist of discrete-valued continuous signals or of events separated by non-uniform periods of time. Some members of the consortium are actively involved in modeling digital circuits with bi-bounded delays using timed automata. As a specification formalism, numerous extensions of temporal logic have been proposed in order to express the metric time aspects of such behaviors.

A timed extension of regular expressions has been proposed recently and has been shown to be equivalent in expressive power to timed automata [ACM02].

After having investigated the timed extensions of temporal logic we have chosen to base STL on the real-time logic MITL [AFH96], obtained from classical “untimed” temporal logic via the following modifications:

1. The *Next* operators is dropped. In untimed sequences, *Next* refers to the next logical time instant, which is, of course, meaningless in the dense order over the reals.
2. The other temporal modalities are replaced by timed temporal modalities which have an interval $[l, m]$ (with l and m integers such that $l < m$) as a parameter. This interval restricts the time variables in the standard semantics of these operators to be within this interval. For example, in the untimed formula *eventually* p , we say that there exists some time t in the future such that p holds at t . In the timed version of this formula we add the constraint that $t \in [l, m]$ relative to the present.

Much of the effort in this work-package was directed toward studying the properties of this logic and the development of monitoring procedures for checking whether a signal satisfies such a formula.

5.1.3 Combining Infinite State-Space with Dense Time

The semantics of $\text{STL}(\mathcal{P})$ is defined in two phases. The initial formula φ is constructed from the state constraints in \mathcal{P} and the Boolean and temporal operators. Then this formula is transformed into an MITL formula φ' by replacing each μ_i with a propositional variable p_i . In parallel, the signal s to be checked is transformed into a Boolean signal s' in which p_i is considered as true in s' at time t if μ_i is satisfied by $s[t]$. Hence s satisfies the original STL formula φ iff s' satisfies the MITL formula φ' .

5.2 Formal Definitions

In this section we formalize the proposed logic and its semantic domain. While doing so we take into account the major validation method that we develop for this logic, namely monitoring. In the presentation we use the mathematical notation for the temporal operators which can be easily transformed into the actual syntax of PSL.

5.2.1 Signals

Let the time domain \mathbb{T} be the set $\mathbb{R}_{\geq 0}$ of non-negative real numbers. A finite length signal s over a domain \mathbb{D} is a partial function $s : \mathbb{T} \rightarrow \mathbb{D}$ whose domain of definition is the interval $I = [0, r)$, $r \in \mathbb{Q}_{\geq 0}$. We say that the length of the signal is r and denote this fact by $|s| = r$. We use the notation $s[t] = \perp$ for every $t \geq |s|$.

Signals over different domains can be combined and separated using the standard pairing and projection operators as well as any pointwise operation. Let $s_1 : \mathbb{T} \rightarrow \mathbb{D}_1$,

$s_2 : \mathbb{T} \rightarrow \mathbb{D}_2$, $s_{12} : \mathbb{T} \rightarrow \mathbb{D}_1 \times \mathbb{D}_2$ and $s_3 : \mathbb{T} \rightarrow \mathbb{D}_3$ be signals and let $f : \mathbb{D}_1 \times \mathbb{D}_2 \rightarrow \mathbb{D}_3$ be a function. The pairing function is defined as

$$s_1 \parallel s_2 = s_{12} \text{ if } \forall t \ s_{12}[t] = (s_1[t], s_2[t]).$$

and its inverse operation, projection as:

$$s_1 = \pi_1(s_{12}) \quad s_2 = \pi_2(s_{12}).$$

The lifting of f to signals is defined as

$$s_3 = f(s_1, s_2) \text{ if } \forall t \ s_3[t] = f(s_1[t], s_2[t]).$$

Note that if s_1 and s_2 differ in length, the convention $f(x, \perp) = f(\perp, x) = \perp$ guarantees that $|s_3| = \min(|s_1|, |s_2|)$.

In the rest of this document, unless otherwise stated, we restrict our attention to Boolean signals, $\mathbb{D} = \mathbb{B}$. In this case (and for discrete domains in general) all reasonable signals are piecewise-constant and can be represented by their values on a countable number of intervals. An *interval covering* for an 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 be *consistent* with a signal s if $s[t] = s[t']$ for every t, t' belonging to the same interval I_i . In that case we can abuse notation and write $s(I_i)$. We say that a signal s is of *finite variability* if it has a finite interval covering. It is not hard to see that such signals are closed under pointwise operations, pairing and projection. We restrict ourselves to signals of finite variability which are, by definition, non-Zeno. 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 $I \subseteq I'$. Clearly, if \mathcal{I}' is consistent with s , so is \mathcal{I} .

We denote by \mathcal{I}_s the minimal interval covering consistent with a finite variability signal s . The set of positive intervals of s is $\mathcal{I}_s^+ = \{I \in \mathcal{I}_s : s(I) = 1\}$ and the set of negative intervals is $\mathcal{I}_s^- = \mathcal{I}_s - \mathcal{I}_s^+$. A Boolean signal $s : \mathbb{T} \rightarrow \mathbb{B}$ can be represented by the pair $(|s|, \mathcal{I}_s^+)$. Such a signal is said to be *unitary* if \mathcal{I}_s^+ is a singleton. Clearly any Boolean signal s of finite variability can be written as $s = s_1 \vee s_2 \vee \dots \vee s_k$ where all s_i are unitary and the boundaries of their corresponding positive intervals do not intersect.

5.2.2 Real-time Temporal Logic

We consider the logic $\text{MITL}_{[a,b]}$ as a fragment of the real-time temporal logic MITL, introduced in [AFH96], such that all temporal modalities are restricted to intervals of the form $[a, b]$ with $0 \leq a < b$ and $a, b \in \mathbb{Q}_{\geq 0}$. More on various dialects of real-time logic can be found in [AH92, Hen98]. The use of bounded temporal properties is justified by the nature of monitoring where the behavior of a system is observed for a finite time interval. Hence unbounded temporal properties are avoided since they may have an ambiguous meaning when monitoring finite behaviors. The basic formulae of $\text{MITL}_{[a,b]}$ are defined by the grammar

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

where p belongs to a set $P = \{p_1, \dots, p_n\}$ of propositions. From basic MITL_[a,b] operators one can derive other standard Boolean and temporal operators, in particular the time-constrained *eventually* and *always* operators:

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

We interpret, MITL_[a,b] over n -dimensional Boolean signals. The satisfaction relation $(s, t) \models \varphi$, indicating that signal s satisfies φ starting from position t , is defined inductively as follows:

$$\begin{aligned} (s, t) \models p & \leftrightarrow \pi_p(s)[t] = \text{T} \\ (s, t) \models \neg\varphi & \leftrightarrow (s, t) \not\models \varphi \\ (s, t) \models \varphi_1 \vee \varphi_2 & \leftrightarrow (s, t) \models \varphi_1 \text{ or } (s, t) \models \varphi_2 \\ (s, t) \models \varphi_1 \mathcal{U}_{[a,b]}\varphi_2 & \leftrightarrow \exists t' \in [t+a, t+b] (s, t') \models \varphi_2 \text{ and} \\ & \quad \forall t'' \in [t, t'], (s, t'') \models \varphi_1 \end{aligned}$$

Note that our definition of the semantics of the time-bounded *until* operator differs slightly from its conventional definition since it requires a time instant $t' \in [t+a, t+b]$ where *both* $(s, t') \models \varphi_2$ and $(s, t') \models \varphi_1$. This definition does not have any repercussion on the derived *eventually* and *always* operators which retain their usual semantics:

$$\begin{aligned} (s, t) \models \diamond_{[a,b]}\varphi & \leftrightarrow \exists t' \in t + [a, b] (s, t') \models \varphi \\ (s, t) \models \square_{[a,b]}\varphi & \leftrightarrow \forall t' \in t + [a, b] (s, t') \models \varphi \end{aligned}$$

A signal s satisfies the formula φ iff $(s, 0) \models \varphi$.

According to the standard semantics for temporal logic, the satisfaction of a formula with unbounded modalities can rarely be determined with respect to a finite signal or sequence. In fact, only the satisfaction of $\diamond p$ or the violation of $\square p$ can be detected in finite time. By using bounded modalities we avoid the problems related to the ambiguity of \models when applied to finite signals or sequences. Nevertheless, even for MITL_[a,b] certain signals are too short to determine satisfaction of the formula, for example the property $\square_{[a,b]}\diamond_{[c,d]}p$ cannot be evaluated on signals shorter than $b+d$. Hence we restrict ourselves to signals which are sufficiently long. The necessary length associated with a formula φ , denoted by $\|\varphi\|$, is defined inductively on the structure of the formula:

$$\begin{aligned} \|p\| & = 0 \\ \|\neg\varphi\| & = \|\varphi\| \\ \|\varphi_1 \vee \varphi_2\| & = \max(\|\varphi_1\|, \|\varphi_2\|) \\ \|\varphi_1 \mathcal{U}_{[a,b]}\varphi_2\| & = \max(\|\varphi_1\|, \|\varphi_2\|) + b \end{aligned}$$

The reader can verify that $s \models \varphi$ is well defined whenever $|s| > \|\varphi\|$.

5.2.3 Real-Valued Signals and STL

In this section we extend our semantic domain and logic to real-valued signals to obtain a dense-time variant of first-order temporal logic. While Boolean signals of finite variability admit a finite representation, this is typically not the case for real-valued signals which are often represented via sampling, that is a sequence of time stamped values of

the form $(t, s[t])$. Although we define the semantics of the logic in terms of the mathematical objects, signals of the form $s : \mathbb{T} \rightarrow \mathbb{R}^m$, we cannot ignore issues related to their effective representation based on the output of some numerical simulator.

Our logic, to be defined in the sequel, does not speak about continuous signals *directly* but rather via a set of *static abstractions* of the form $\mu : \mathbb{R}^m \rightarrow \mathbb{B}$. Typically μ will partition the continuous state-space according to the satisfaction of some inequality constraints on the real variables. As long as $\mu(s[t])$ remains constant we do not really care about the exact value of $s[t]$. However, in order to evaluate formulae we need the sampling to be sufficiently dense so that all such transitions can be detected when they happen. The problem of “event detection” in numerical simulation is well-known and can be resolved using variable step adaptive methods for numerical integration.

However this may raise problems related to finite variability and Zenoness (infinitely many state transitions in a bounded interval of time). Consider an abstraction $\mu : \mathbb{R} \rightarrow \mathbb{B}$ defined as $\mu(x) = 1$ iff $x > 0$ and consider a signal s that oscillates with an unbounded frequency around the origin. Such a signal will cross zero too often and its abstraction may lead to Boolean signals of infinite variability. These are eternal problems that need to be solved pragmatically according to the context. In any case the dynamics of most reasonable systems have a bounded frequency, and even if we add white noise to a system, the frequency remains bounded by the size of the integration step used by the simulator. From now on we assume that we deal with signals that are well-behaving with respect to every μ , that is, $\mu(s)$ has a bounded variability and every change in $\mu(s)$ is detected in the sense that every point t such that $\mu(s[t]) \neq \lim_{t' \rightarrow t} \mu(s[t'])$ is included in the sampling.

Definition 1 (Signal Temporal Logic) *Let $\mathcal{P} = \{\mu_1, \dots, \mu_n\}$ be a collection of constraints, effective functions of the form $\mu_i : \mathbb{R}^m \rightarrow \mathbb{B}$. An STL(\mathcal{P}) formula is an MITL $_{[a,b]}$ formula over the atomic propositions $\mu_1(x), \dots, \mu_n(x)$.*

Any signal which is well-behaving with respect to \mathcal{P} can be transformed into a Boolean signal $s' : \mathbb{T} \rightarrow \mathbb{B}^n$ such that $s' = \mu_1(s) \parallel \mu_2(s) \parallel \dots \parallel \mu_n(s)$ is of bounded variability. By construction, for every signal s and STL formula φ , $s \models \varphi$ iff $s' \models \varphi'$ in the MITL $_{[a,b]}$ sense where φ' is obtained from φ by replacing every $\mu_i(x)$ by a propositional variable p_i .

The process of checking satisfaction of an STL formula φ by a signal s decomposes into two parts. As a first step we construct a Boolean “filter” for every $\mu_i \in \mathcal{P}$ which transforms s into a Boolean signal $p_i = \mu_i(s)$. The signal thus obtained can be checked for satisfaction against the corresponding MITL $_{[a,b]}$ formula φ' .

To illustrate the Boolean filtering process, consider the signal $\sin[t]$ where t is given in degrees and $\mu(x) = x > 0$. The signal is of length 400 and is sampled every 50 time units plus two additional sampling points to detect zero crossing at 180 and 360. The input to the Boolean filter is

$$(0, 0.0), (50, 0.766), (100, 0.984), (150, 0.5), (180, 0.0), (200, -0.342), \\ (250, -0.939), (300, -0.866), (350, -0.173), (360, 0), (400, 0.643)$$

and the output is a Boolean signal p such that $\mathcal{I}_p^+ = \{[0, 180), [360, 400)\}$.

6 Some Examples

In this section we demonstrate some typical properties that can be expressed in STL and give examples of their satisfaction and violation by signals.

6.1 Following a Reference Signal

As a first example we show how the fact that a signal follows another signal with some delay can be expressed in STL. We consider two periodic signals x_1 and x_2 , ranging in $[-1, +1]$ and want to express the property that whenever one of them crosses the threshold of 0.7, so does the other within $t \in [3, 5]$ time units. The corresponding property is:

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

Let us fix the first signal 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. Figure 1 shows the two signals where x_2 is generated with negligible θ and hence its form is close to that of x_1 . The Boolean signals p_1 and p_2 are the Boolean abstraction of x_1 and x_2 , respectively, via the constraint $x > 0.7$. In other words, they show the points in time when the signals satisfy this constraint. The monitoring procedure generates Boolean signals for all sub-formulae, for example the signal $\Diamond_{[3,5]}(x_2 > 0.7)$ is high at all points in time t for which $x_2 > 0.7$ at time $t + t'$ with $t' \in [3, 5]$. The last signal in Figure 1 shows the truth value of the main formulae over time, and since it is true at time zero, the signals satisfy the property.

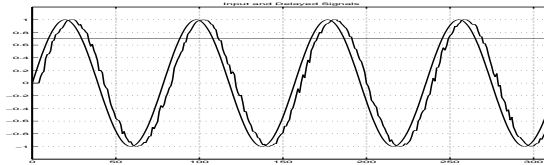
In Figure 2) we have chosen to generate x_2 with much larger additive noise $\theta \in [-0.5, 0.5]$. The fluctuations in the value of x_2 are reflected in the Boolean abstraction p_2 and lead to a violation of the property at some points where $x_1 > 0.7$ is not followed by $x_2 > 0.7$ within the pre-specified time.

6.2 Stabilizability

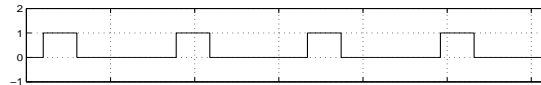
The second example is a very typical stabilizability property used extensively in control and signal processing. The system in question is a “plant” has to maintain an output signal around a fixed level despite disturbances from the outside world. The actual system used to generate this example is a water-level controller for a nuclear plant. The disturbances come from changes in the system load that trigger changes in the operations of the reactor that change the water level. The role of the control system is to stabilize the water level again around the desired value.

This property(Figure 4) we see that the output signal y violates the property both by over-shooting below -30 and by taking more than 150 time units to return to $[-0.5, 0.5]$.

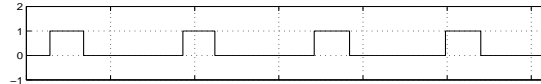
$$s = x_1 || x_2$$



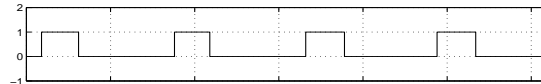
$$p_1 = x_1 > 0.7$$



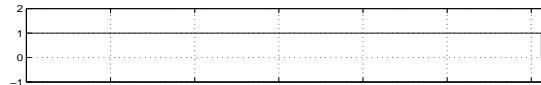
$$p_2 = x_2 > 0.7$$



$$\diamond_{[3,5]} p_2$$



$$p_1 \rightarrow \diamond_{[3,5]} p_2$$

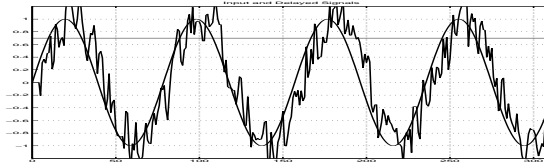


$$\square_{[0,300]} (p_1 \rightarrow \diamond_{[3,5]} p_2)$$

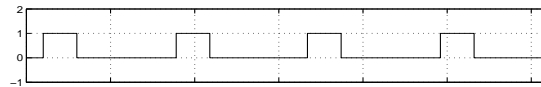


Figure 1: A 2-dimensional signal satisfying the property $\square_{[0,300]}((x_1 > 0.7) \Rightarrow \diamond_{[3,5]}(x_2 > 0.7))$. Boolean signals correspond to the evolution of the truth values of sub-formulae over time.

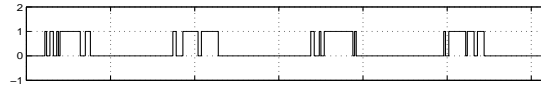
$$s = x_1 \parallel x_2$$



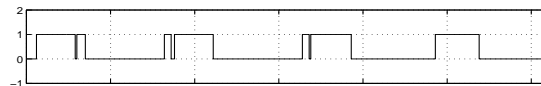
$$p_1 = x_1 > 0.7$$



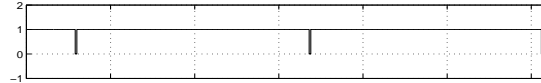
$$p_2 = x_2 > 0.7$$



$$\diamond_{[3,5]} p_2$$



$$p_1 \rightarrow \diamond_{[3,5]} p_2$$



$$\square_{[0,300]} (p_1 \rightarrow \diamond_{[3,5]} p_2)$$

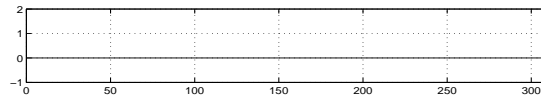


Figure 2: A 2-dimensional signal violating the property $\square_{[0,300]}((x_1 > 0.7) \Rightarrow \diamond_{[3,5]}(x_2 > 0.7))$.



Figure 3: A disturbance signal and an analog response y satisfying the stabilizability property $\square_{[300,2500]}((|y| \leq 30) \wedge ((|y| > 0.5) \Rightarrow \diamond_{[0,150]}\square_{[0,20]}(|y| \leq 0.5)))$.

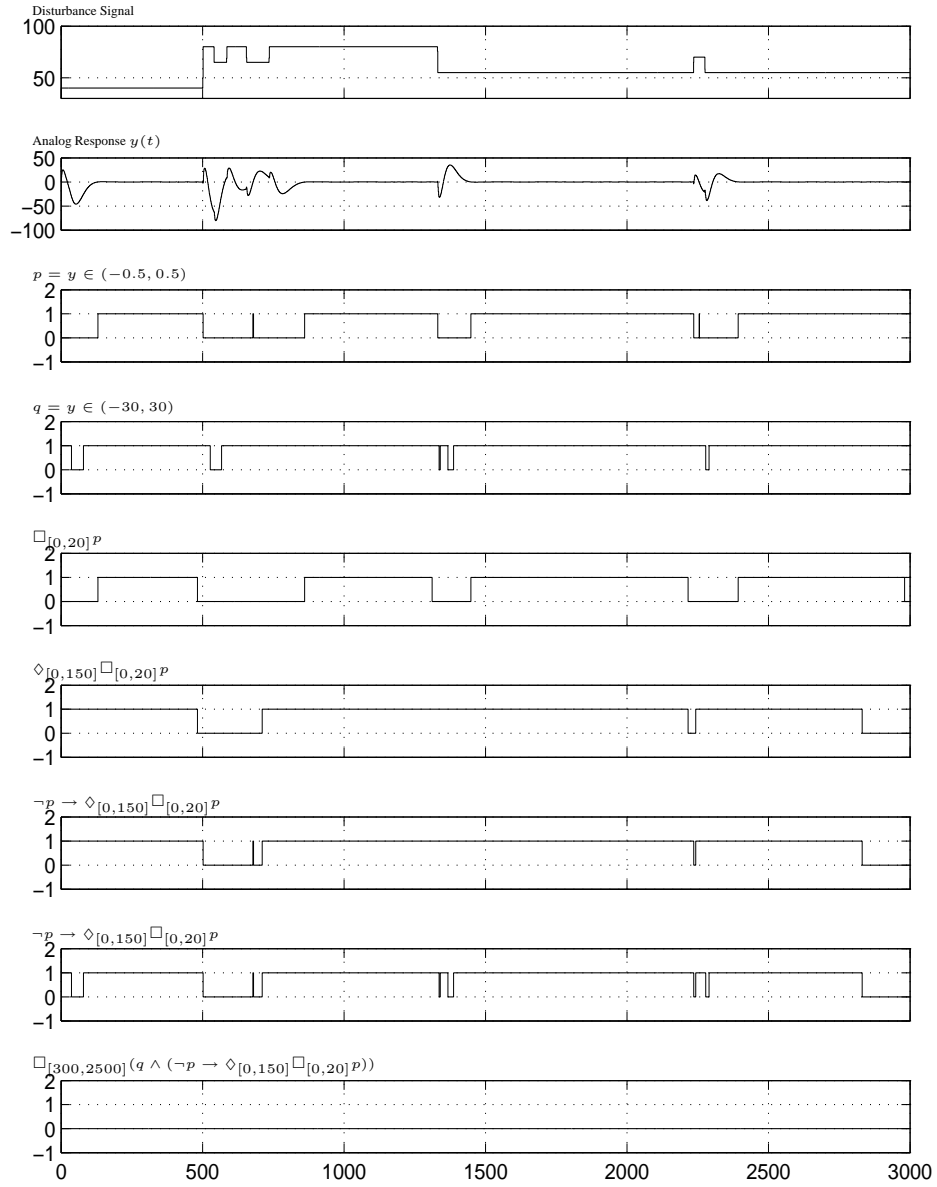


Figure 4: A disturbance signal and an analog response y violating the stabilizability property $\Box_{[300,2500]} ((|y| \leq 30) \wedge ((|y| > 0.5) \Rightarrow \Diamond_{[0,150]} \Box_{[0,20]} (|y| \leq 0.5)))$.

7 Potential Extensions

In this section we mention several possible extensions of the proposed logic, some of which are subject to ongoing work to be reported in Deliverable D1.3/2.

7.1 Simple Extensions

Here we mention extensions that may add some expressive power but are based on the same principles as STL and express the same type of properties.

7.1.1 Events

MITL and STL are state-based formalisms and as such they cannot distinguish a point in time where p *becomes* true from other points where p is simply true. Using primitive such as $p \uparrow$ and $p \downarrow$ for the rising and falling of p , one can express properties such as bounded variability (the distance between any two successive changes is at least d) as:

$$\Box_{[0,r]}((p \uparrow \Rightarrow \Box_{[0,d]}p) \wedge (p \downarrow \Rightarrow \Box_{[0,d]}\neg p)).$$

There is no problem in adding this feature to the logic except for slightly complicating the monitoring procedure due to the need to consider all combinations of left/right closed/open intervals.

7.1.2 Past

Past operators do not add expressive power to temporal logic but sometimes facilitate the expression of certain properties. One advantage they have over future operators is that the construction of monitors for past formulae in a form of deterministic timed automata is much simpler (this is a new result obtained by consortium members and will be discussed in Deliverable D3.2/6). The semantic definition of past operators is symmetric to those of future operators.

7.1.3 Regular Expressions

Adding timing constraints to regular expressions is done in a different style than in temporal logic. The timing restriction is realized by the $\langle \varphi \rangle_I$ operator which constrains the metric length (duration) of the signals satisfying φ to be in an integer-bounded interval I . We sketch here the syntax and the semantics of timed regular expressions as introduced in [ACM02]. A variant of this formalism, adapted to real-valued and multi-dimensional signals, will be presented in deliverable D1.3/2.

The set $\mathcal{E}(\Sigma)$ of timed regular expressions over an alphabet Σ , is defined recursively as either a , $\varphi_1 \cdot \varphi_2$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$, φ^* or $\langle \varphi \rangle_I$ where $a \in \Sigma$, $\varphi, \varphi_1, \varphi_2 \in \mathcal{E}(\Sigma)$ and I is an integer-bounded interval. We use the notation a^r for a signal whose value is constantly a and its metric length (duration) is r . The semantics of timed regular expressions shares common features with standard regular expressions in what concerns

Boolean operations, concatenation and Kleene star. The two specific features are the semantics of the atomic symbols, defined as

$$\llbracket a \rrbracket = \{a^r : r \in \mathbb{R}_+\},$$

that is, the set of all a -signals of arbitrary duration, and the semantics of the time restriction operator which is

$$\llbracket \langle \varphi \rangle_I \rrbracket = \cap \{s : s \in \llbracket \varphi \rrbracket \wedge |s| \in I\}.$$

Here are some examples of expressions and their semantics:

- Expression $\langle a \rangle_{(0,3]}$ denotes the set of a signals of length in $(0, 3]$.
- Expression $\langle a \cdot b \rangle_{(0,3]}$ denotes all signals composed of an a part followed by a b part where the sum of duration of the two parts is in $(0, 3]$.
- Expression $\langle \langle a \cdot b \rangle_{(0,3]}^* \rangle$ denotes all signals composed of zero or more repetitions of elements of $\llbracket \langle a \cdot b \rangle_{(0,3]} \rrbracket$.
- Expression $\langle \langle a \cdot b \rangle^* \rangle_{(0,3]}$ denotes all signals of total duration in $(0, 3]$, consisting of zero or more repetitions of ab signals.
- Expression $\langle a \cdot b \rangle_3 \cdot c \wedge a \cdot \langle b \cdot c \rangle_3$ denote all signals of the logical form abc such that c starts 3 time units after the beginning of the signal and ends 3 time units after b starts.

7.2 Richer Temporal Properties

All properties discussed so far are based on static abstractions from the continuous state-space into Booleans. This means that values of a signal at different time instants can “communicate” only through their Boolean abstractions. For example we can define constraints on the temporal distance between a point t where $x[t] > c$ and a point t' where $x[t'] < d$ but we can say nothing in STL about the difference $x[t] - x[t']$. To be able to speak of such properties we will need to extend the logic with non-Boolean “filters”, that is, operators that transform real-valued signals into other real-valued signals. Such operators can be memoryless like point-wise arithmetic operations, or operators with memory such as integrators. The construction of monitors for this type of properties will have to rely on blocks for such operators that exist in the respective simulation tools.

7.3 Infinitary Properties

Temporal logic is traditionally interpreted over infinite sequences for which it can express infinitary ω -properties that specify repeatedly occurring patterns and periodic behaviors. Since our work is geared toward monitoring which works, by definition, on finite signals, we will not consider such extensions at this point. Life is difficult enough with continuous signal of finite duration.

7.4 Non-Temporal Properties

From discussions with designers it turns out that frequency-domain properties of signals are very useful for evaluating certain circuits. Such properties are quite different from time-domain properties and we hope, toward the end of the project, to propose some formal support for them. Monitoring for such properties will be based on running the signal through time-to-frequency transforms, such as Fourier's, and checking the properties of the obtained spectrum.

8 Additional Related Work

This section provides some additional information that allows the reviewers to situate the work on language extension in the general context of the analog activities of PROSYD.

8.1 Monitoring

Monitoring is not part of the current deliverable and will be presented later in Deliverable D3.2/6. However we find it useful to report the progress made in this direction, because this is the ultimate application of the proposed language and hence its major adequacy criterion. So far we have developed and implemented a monitoring procedure for STL [MN04]. This procedure reads a property and a real-valued signal, transforms it into a Boolean signal and, through backward propagation of truth values, establishes satisfaction. This procedure, due to its backward nature, can only be applied offline, that is after the simulation has terminated. A prototype implementation of such a monitoring procedure has been interfaced with Matlab/Simulink and used to generate the monitoring examples in Section 6.

We are currently working on the development of an online procedure that can sometimes detect violation or satisfaction of certain formulae based on a prefix of the simulation trace. Such an extension requires some new theoretical results about the determinization of timed automata.

8.2 Verification

In parallel with the work on monitoring, we continue with our efforts to push the limits of exhaustive verification of analog circuits. In the paper attached to this deliverable as an appendix [DDM04] we apply verification techniques to two analog circuits. We use reachability analysis techniques for hybrid systems to verify a Biquad low-pass filter and another technique, based on bounded-horizon optimal control, to show that a Sigma-Delta modulator (an important ingredient of analog to digital converters) does not reach a saturation point.

8.3 Workshop Organization

Members of the consortium, together with other academic and industrial colleagues organize the first workshop on verification of analog circuits to be held in Edinburgh on April 2005. It is hoped that such a forum will increase the awareness of analog designers to the potential contribution of formal verification to the design process and will also divert some of the energy of the verification community toward these problems.

9 Conclusions

We have presented the major ingredient of the extension of PSL toward analog circuits, the logic STL that takes into account the particular features of analog signals. This logic is already covered by a monitoring procedure that can check satisfaction of STL formulae by simulation traces. The next steps for the second year of the project are:

1. More discussions with analog designers in order to get some feed-back on the proposed logic and its suitability for describing properties of certain circuits. One difficulty is that analog designers tend to be very busy persons and do not have time to invest in ideas that come from an alien culture. In a recent meeting between Verimag and ST in Agrate it was decided to look at flash memory specifications as a possible test-case. Effort will be made to meet more designers from other ST sites.
2. Enriching the logic with some of the features mentioned in Section 7 with priority to adding events, richer temporal properties and treatment of regular expressions. Some of the decisions will be based on feedback from designers.
3. Improving the monitoring procedure to work as online as possible and to treat the extensions to the logic.

References

- [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(1):116–146, 1996.
- [AH92] R. Alur and T.A. Henzinger. Logics and Models of Real-Time: A Survey. In *Proc. REX Workshop, Real-time: Theory in Practice*, pages 74–106. LNCS 600, Springer, 1992.
- [ACM02] E. Asarin, P. Caspi and O. Maler, Timed Regular Expressions *The Journal of the ACM* 49, 172-206, 2002.
- [ADM02] E. Asarin, T. Dang, and O. Maler. The d/dt tool for verification of hybrid systems. In *Computer Aided Verification*, LNCS 2404, 365–370, Springer, 2002.
- [DDM04] T. Dang, A. Donze and O. Maler, Verification of Analog and Mixed Signal Circuits using Hybrid Systems Techniques, *Proc. FMCAD'04*, 2004.
- [Hen98] T.A. Henzinger. It's about Time: Real-time Logics Reviewed. In *Proc. CONCUR'98*, pages 439–454. LNCS 1466, Springer, 1998.
- [MN04] O. Maler and D. Nickovic, Monitoring Temporal Properties of Continuous Signals, *proc. FORMATS/FTRTFT'04*, LNCS 3253, 2004.
- [MP03] O. Maler and A. Pnueli (eds). *Hybrid Systems: Computation and Control*. LNCS 2623, Springer, 2003.
- [T03] S. Tripakis, Folk Theorems on the Determinization and Minimization of Timed Automata, *Proc. FORMATS'03*, 2003.
- [VW86] M.Y. Vardi and P. Wolper. An Automata-theoretic Approach to Automatic Program Verification. In *Proc. LICS'86*, pages 322–331. IEEE, 1986.