FP6-IST-507219

PROSYD:

Property-Based System Design

Instrument: Specific Targeted Research Project

Thematic Priority: Information Society Technologies

Final Proposal for PSL Analog Extensions (Deliverable 1.3/2)

Due date of deliverable: January 1, 2007
Actual submission date: January 1, 2007

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	☒
PP	Restricted to other programme participants (including the Commission Services)	□
RE	Restricted to a group specified by the consortium (including the Commission Services)	□
CO	Confidential, only for members of the consortium (including the Commission Services)	□

Notices

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

This document is intended to fulfil the contractual obligations of the PROSYD project concerning deliverable 1.3/2 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.

Table of Revisions

Version	Date	Description and reason	By	Affected sections
0.1	November 15, 2006	First draft	Nickovic	All
0.5	December 28, 2006	Updated and corrected version	Nickovic	All
1.0	December 31, 2006	Final Version	Nickovic	All

Authors

Dejan Nickovic
Oded Maler
Amir Pnueli
Paul Caspi
Antoine Girard

Executive Summary

This document presents the final proposal for an extension of PSL toward real-time and analog properties. It is based on the results described in Deliverable 1.3/1 [M05] and on the feedback from the analog designers of ST Microelectronics Italy. The resulting STL/PSL language allows to specify properties of analog and mixed-signal systems. STL/PSL logic is mainly intended to be used for *lightweight verification* of such properties.

Purpose

The purpose of this document is to descibe the efforts of defining a language based on temporal logic and in the spirit of PSL that will be the foundation of expressing properties describing behaviors of analog and mixed signal systems.

Intended Audience

This document is intended for formal methods researchers who are particularely interested in the analysis of timed and analog systems. It will provide them with a formal specification language for expressing properties on continuous signals. This document is also intended to designers familiar with PSL that are interested in validating analog circuits. It is a pioneering work and a first step in trying to bring closer the analog and digital communities.

Background

This document is the continuation of the Deliverable 1.3/1 [M05]. The theoretical background is mainly inspired by the results of [AFH96] and [MN04]. The distance-based operators included in the final proposal of STL/PSL are adapted from the work done in [FGP06, KC06a, KC06b]. This document is also related to Deliverable [NM06a] which presents the *lightweight verification* tool based on STL/PSL and to Deliverable 3.4/2 [NM06b] which describes a case study on monitoring Flesh memory simulations against specifications written in STL/PSL.

Contents

Table of Figures

Glossary

Exhaustive verification

The process of proving the correctness of a system with respect to a formal specification by exhaustively exploring its mathematical model.

Leightweight verification

The procedure for proving the correctness of a single finite execution of a system with respect to a formal specification.

LTL

Linear-time Temporal Logic

MITL

Metric Interval Temporal Logic. The dense-time extension of the LTL logic, allowing modalities ranging over a non-punctual interval

Monitoring

See **lightweight verification**.

PSL

Property Specification Language

STL

Signal Temporal Logic. The analog extension of MITL

STL/PSL

Signal Temporal Logic/Property Specification Language. The analog extension of PSL

1 Introduction

This document is the final proposal for an extension of PSL [KCV04, HFE04] toward real-time and analog properties and is the continuation of the Deliverable 1.3/1 [M05]. The extension of temporal logics to richer semantic domains is a recent direction of research in the field. It has been studied in [ASS+05] in the context of run-time verification of synchronous software with complex data structures. We can also mention using timed temporal logics in exhaustive verification of continuous systems[FGP06].

The results described in this document represent mainly the work of Verimag and Weizmann. The additional participation of ST on providing information about analog design of circuits was crucial in identifying the useful properties that describe analog behavior and enriching the language according to the industrial needs.

1.1 Relation to Deliverable 1.3/1

The final proposal for an analog extension of PSL [KCV04, HFE04] relies on the report described by the Deliverable 1.3/1 [M05]. However, the proposed final extension differs from the preliminary report in the following aspects:

1. Deliverable 1.3/1 included more options (such as regular expressions, and preliminary ideas on frequency domain properties) that finally did not make their way to the final proposal which is restricted to signal temporal logic (STL) [MN04] and some metric constructs. The reason for this omission is that we prefer to restrict the proposal to language constructs that have been found useful for the treatment of the analog case study, and for which we had feedback from designers. The project proposal was written with ST Crolles as a partner which was supposed to give us more feedback from analog designers but since the responsibility to this part has shifted to ST Italy, we had less frequent contacts with designers and found no use to define hypothetical extensions. Consequently we focused on STL, as the natural extension of the temporal logic part of PSL for expressing sequential properties of analog signals.
2. While in Deliverable 1.3/1 the extensions were described in a purely mathematical style, the present document follows the style and the syntax of PSL and is richer in explanations and illustrations.

3. The metric constructs presented in this document (and implemented into our monitoring tool, see Delivrable 3.2/13 [NM06a]) which are used to measure the similarity between an analog signal and a reference signal were not included in Deliverable 1.3/1 and appear here for the first time.

1.2 Overview

PSL [KCV04, HFE04] consists of four layers shown in Figure 1 (a): Boolean, temporal, verification and modeling. The Boolean layer consists of Boolean expressions that are used by the other layers, mainly the temporal one. The temporal layer is the heart of the PSL language and is used to express complex temporal relations between Boolean signals. In PSL, temporal expressions are evaluated over cycles (discrete steps). The verification layer consists of directives telling the verification tools what to do with the properties described in the temporal layer. Finally, the modeling layer is used to model the behavior of the design under verification.

Analog systems are different from the discrete system in several aspects. An important difference is that variables describing the dynamics of an analog system are real-valued. Unlike digital systems whose dynamics evolve in cycles, which are discretized time samples, the analog systems have continuous dense-time behavior. The effort of extending the expressiveness of PSL in order to adapt its semantics to such real-valued dense-time behavior has been conducted on several different axes shown in Figure 1 (b):

- Adding an *analog layer* which deals with real-valued variables to the current structure of PSL
- Building a “bridge” between the analog and the temporal layers
- Identifying a proper subset of the temporal layer of PSL and adapting it to the dense-time

The modeling, verification and Boolean layers can be used in the analog extension of PSL. However, from now on, we will concentrate only on the parts of the language that have to be adapted to the new semantics, that is the embedding of the analog layer and the Boolean abstraction to PSL, as well as the necessary adaptation of the temporal layer to support the dense time semantics. We call the resulting extended logic STL/PSL.

A basic notion in STL/PSL is that of a *signal*, a partial function from time to some arbitrary domain (depending on the type of the signal). Signals are defined formally in the next Section. Each operator in STL/PSL has a *representative signal*

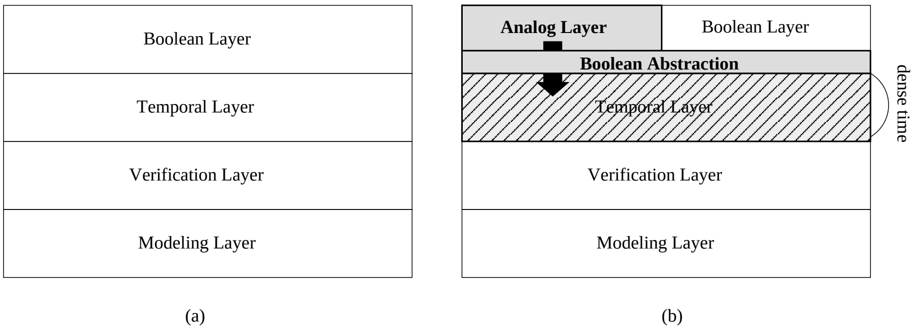


Figure 1: (a) PSL layers (b) STL/PSL layers

associated to it. For the operators in the analog layer, we call them *continous signals*, and they correspond to the result of applying the operator to the input at different time instants. The *satisfaction signal* associated to temporal and Boolean subformulae shows at each time instant whether the subformula is satisfied or not at that time point.

The analog layer of STL/PSL consists of expressions that represent the analog design. It is shaped around built-in functions that manipulate real-valued variables. Such functions allow expressing analog transformations such us derivatives or absolute values of such variables. Note that the analog layer cannot be used on its own to validate an STL/PSL property. The continuous behavior described in the analog layer needs first to be abstracted before relating the temporal behavior between different analog components.

The main objective of the PSL analog extension is to add constructs that relate behaviors of analog variables at different points in time by applying standard operators from the PSL temporal layer. In order to achieve this goal we need to abstract the values of analog inputs. The first method that we propose is the Boolean abstraction of real valued variables with *thresholds*, and is at the heart of the STL logic presented in [MN04]. Note that the Boolean abstraction based on inequality contraints is already included in the Boolean layer of PSL. Given that it represents an important feature of STL/PSL we will treat independently such abstractions in this document.

The other technique that we use to bridge the analog to the temporal layer are *metric distances* between analog variables. The attractiveness of the distance-based operators comes from their *metric* nature, from which we can extract richer quantitative information about the degree of similarity between two continuous signals. The distance operators will be mainly used for comparison of simulation traces with respect to a reference (expected) output.

The temporal operators in STL/PSL are applied to the Boolean abstractions of the continuous signals or directly to the Boolean (satisfaction) signals. For continuous signals, the original values are lost in this process, but the timing information of the Boolean abstraction remains dense. Hence, interpreting PSL over clock cycles is not adapted to this framework. Regular expressions, LTL and CTL are the subsets of PSL that can be naturally extended to real time, using the *timed regular expressions* [ACM02], MITL [AFH96, MN04, MNP06] and TCTL [Y97] real time formalisms. TCTL, the real time version of CTL is a branching-time logic, and as such is not well adapted to describe temporal behaviors of individual traces, which is the primary purpose of STL/PSL. Combining timed regular expressions and MITL in the timed framework is difficult due to the difference in defining the input alphabets. From our experience, MITL is the most natural formalism for expressing real-time properties and has been chosen as the basis for the temporal layer of STL/PSL.

Finally, STL/PSL allows combining analog and Boolean variables in a single property, and hence can also be used to express properties of mixed-signal designs.

This document is structured as follows:

- Section 2: presents the theoretical background behind the STL/PSL logic
- Section 3: Describes the effort of defining concretely the syntax and semantics of STL/PSL, adapting the syntax to the one of PSL. It defines the analog layer of STL/PSL and its operators, the Boolean abstraction part of the language that allows “bridging” the analog and the temporal layers of STL/PSL, the real-time extensions of the temporal layer of PSL, and embedding metric distance-based operators into the language.
- Section 4: Conclusions.
- Appendix A: Production rules for STL/PSL.

2 Theoretical Background

This Section presents the technical definitions behind the STL/PSL language.

2.1 Signals

A signal ξ defined over an arbitrary domain $\mathbb{D}$ is a partial function $\xi : \mathbb{R}_{\geq 0} \rightarrow \mathbb{D}$. Signal ξ is said to be of *finite length* if its domain of definition is the interval $\mathbb{T}_\xi = [0, r)$. The length of ξ is denoted by $|\xi| = r$ and we use the notation $\xi[t] = \perp$ when $t \geq |\xi|$. Signals can be combined and separated using the standard operations of *pairing* and *projection* defined as

$$\begin{gathered} \xi_1 \,||\, \xi_2 = \xi_{12} \text{ if } \forall t\ \xi_{12}[t] = (\xi_1[t], \xi_2[t]) \\ \xi_1 = \pi_1(\xi_{12}) \ \ \xi_2 = \pi_2(\xi_{12}) \end{gathered}$$

We are interested in two particular types of signals, Boolean signals $\xi_b : \mathbb{R}_{\geq 0} \rightarrow \mathbb{B}$ and continuous signals $\xi_a : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$.

A Boolean signal is a sequence of left-closed right-open intervals $\xi_b = I_0 \cdot I_1 \cdot \ldots, I_k$ such that $I_0 = [0, t_1)$, $I_i = [t_i, t_{i+1})$, $I_k = [t_{k-1}, r)$ and the value of ξ_b is constant in every interval. The definition of Boolean signals using left-closed right-open disallows punctual intervals, in other words for any $I_p = [t, t)$ is empty.

2.2 Real-time logic MITL

The temporal layer of the STL/PSL specification language is based upon the real-time logic MITL [AFH96] interpreted over dense Boolean signals. MITL is a natural real-time extension of the LTL logic supported by PSL. The principal modality of MITL is the timed until $\mathcal{U}_I$ where I is some non-singular interval. A formula $p \mathcal{U}_{[a,b]} q$ is satisfied by a model at any time instant t that admits q at some $t' \in [t + a, t + b]$, and where p holds continuously from t to t'. A consequence of interpreting MITL over dense time is that the *next* operator loses its meaning and hence is not used in the logic.

The basic formulae of MITL are defined by the grammar:

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

MITL formulae are interpreted over n-dimensional Boolean signals. The signal ξ_b is a multi-dimensional Boolean signal containing all propositional Boolean signals p. We use p to denote the signal $\pi_p(\xi_b)$, the projection of the multi-dimensional signal ξ_b on p.

The satisfaction relation $(\xi_b, t) \models \varphi$, indicating that signal ξ_b satisfies φ starting from position t, is defined inductively as follows:

$$\begin{array}{lcl}
(\xi_b,t) \models p & \leftrightarrow & \pi_p(\xi_b)[t] = \mathrm{T} \\
(\xi_b,t) \models \neg\varphi & \leftrightarrow & (\xi_b,t) \not\models \varphi \\
(\xi_b,t) \models \varphi_1 \vee \varphi_2 & \leftrightarrow & (\xi_b,t) \models \varphi_1 \text{ or } (\xi_b,t) \models \varphi_2 \\
(\xi_b,t) \models \varphi_1 \mathcal{U}\, \varphi_2 & \leftrightarrow & \exists t' \geq t \text{ st } (\xi_b,t') \models \varphi_2 \text{ and} \\
 & & \forall t'' \in [t,t'].(\xi_b,t'') \models \varphi_1 \\
(\xi_b,t) \models \varphi_1 \mathcal{U}_{[a,b]} \varphi_2 & \leftrightarrow & \exists t' \in [t+a, t+b]\ (\xi_b,t') \models \varphi_2 \text{ and} \\
 & & \forall t'' \in [t,t'], (\xi_b,t'') \models \varphi_1
\end{array}$$

From the basic MITL operators we can define the *eventually* and *always* operators:

$$\begin{array}{lcl}
\Diamond\varphi & = & \mathrm{T}\,\mathcal{U}\,\varphi \\
\Box\varphi & = & \neg\Diamond\neg\varphi \\
\Diamond_{[a,b]}\varphi & = & \mathrm{T}\,\mathcal{U}_{[a,b]}\varphi \\
\Box_{[a,b]}\varphi & = & \neg\Diamond_{[a,b]}\neg\varphi
\end{array}$$

Each subformula of an MITL property has an associated *representative signal* called the *satisfaction signal* $\xi'_b = \chi_\varphi(\xi_b)$. The signal $\xi'_b = \chi_f(\xi_b)$ satisfies $\xi'_b[t] = 1$ iff $(\xi_b,t) \models \varphi$. Our definition of MITL [MN04] slightly deviates from the original one in [AFH96]:

1. We disallow signals that admit punctuality
2. We restrict modalities to closed intervals
3. We modify the semantics of $p\ \mathcal{U}\ q$ to require a “handshake” moment where both p and q hold

The restriction to non-punctual signals is very reasonable from a semantic point of view and constitutes the natural choice for signals (refer to [ACM02] for the algebraic definition of signals and their properties). The two other modifications are consequences of this choice as we want the signals representing the satisfaction of the temporal subformulae to be valid signals as well. The main limitation of this logic is the inability to specify events (or the rising and falling of a signal) which prevents, for example, expressing properties such as bounded variability [1].

2.3 Signal Temporal Logic

The analog part of STL/PSL relies upon the Signal Temporal Logic (STL), presented in [MN04]. STL extends the MITL logic and its semantic domain to real-valued signals. The analog component of STL is embedded into MITL via *static abstractions* of the form $\mu : \mathcal{R} \to \mathbb{B}$, partitioning the continuous state-space according to the satisfaction of some inequality constraints on the real-valued variables and expressions.

We first define analog expression as:

$$\phi := a \mid f(\phi)$$

where a is an analog variable and $f(\phi)$ is a function transforming real-valued signals into real-valued signals[2] $f : (\mathcal{R}_+ \to \mathcal{R}) \to (\mathcal{R}_+ \to \mathcal{R})$. Analog expressions in ϕ have an associated signal ξ_ϕ called *continuous signal* such that $\xi_\phi[t] = \phi(\xi_a)[t]$.

The syntax of STL is defined by the following grammar:

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

where ϕ is an analog expression, $\circ \in \{<, \leq\}$, and $c \in \mathcal{Q}$ is a constant. Note that STL formulae are interpreted over a union of continuous and Boolean signals $\xi = \xi_a \cup \xi_b$. The satisfiability relation for the static abstraction $\phi \star c$ is

$$(\xi, t) \models \phi \circ c \quad \leftrightarrow \quad \phi[t] \circ c$$

According to the layered approach of STL/PSL, ϕ corresponds to the *analog layer*, $\phi \circ c$ is the *Boolean abstraction* and φ is the *temporal layer*.

[1] However, events can be approximated with arbitrary precision, see D3.2/13 [NM06a]

[2] The abstract definition of f provides a general framework to easily "plug-in" specific built-in analog functions into logics based on STL

2.4 Finitary interpretation of STL

The analog extension of PSL is intended for specification of properties that are to be used for lightweight verification of simulation traces of finite length. Temporal operators may have a termination condition that comes at an interval that occurs after the end of the signal. This implies that the underlying logic has to provide the finitary interpretation of formulae.

We take the approach developed in PSL, providing *strong* and *weak* forms of the temporal operators. The *strong form* requires the terminating condition to occur before the end of the signal, while the *weak form* makes no such requirements. In PSL for example, `until!` and `until` present the strong and the weak forms of the until operator, respectively.

Consequently, we adapt the semantics of STL to finitary traces and call the resulting logic STL^f. STL^f is interpreted over a multidimensional signal $\xi = \xi_a \cup \xi_b$. The temporal operators of STL^f are decorated with *w* and *s* superscripts, denoting their *weak* or *strong* form respectively. We remind that $\mathbb{T}_\xi$ denotes the time interval corresponding to the duration of ξ. The syntax of STL^f is defined by the grammar:

$$\varphi := p \mid \phi \circ c \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \,\mathcal{U}^{s} \varphi_2 \mid \varphi_1 \,\mathcal{U}^{s}_{[a,b]} \varphi_2 \mid \Diamond^{s}\varphi_1 \mid \Diamond^{s}_{[a,b]}\varphi_1 \mid \Diamond^{w}_{[a,b]}\varphi_1$$

Note that we explicitly define different *eventually* operators, since we will use them to obtain timed versions of other operators. The semantics of the temporal operators is defined as follows:

$$\begin{array}{lcl}
(\xi,t) \models \varphi_1 \,\mathcal{U}^{s} \varphi_2 & \leftrightarrow & \exists t' \in [t,\infty) \cap \mathbb{T}_\xi \text{ st } (\xi,t') \models \varphi_2 \text{ and } \forall t'' \in [t,t']\ (\xi,t) \models \varphi_1 \\
(\xi,t) \models \varphi_1 \,\mathcal{U}^{s}_{[a,b]} \varphi_2 & \leftrightarrow & \exists t' \in [t+a,t+b] \cap \mathbb{T}_\xi \text{ st } (\xi,t') \models \varphi_2 \text{ and} \\
(\xi,t) \models \Diamond^{s}\varphi_1 & \leftrightarrow & \exists t' \in [t,\infty) \cap \mathbb{T}_\xi \text{ st } (\xi,t') \models \varphi_1 \\
(\xi,t) \models \Diamond^{s}_{[a,b]}\ \varphi_1 & \leftrightarrow & \exists t' \in [t+a,t+b] \cap \mathbb{T}_\xi\ (\xi,t') \models \varphi_1 \\
(\xi,t) \models \Diamond^{w}_{[a,b]}\ \varphi_1 & \leftrightarrow & \exists t' \in [t+a,t+b] \cap \mathbb{T}_\xi\ (\xi,t') \models \varphi_1 \\
 & & \text{ or } |\xi| \leq t+b
\end{array}$$

From the basic operators, we can derive other ones, and we give some examples:

$$\begin{array}{lcl}
\Box^{w}\varphi & = & \neg\Diamond^{s}\neg\varphi \\
\Box^{w}_{[a,b]}\varphi & = & \neg\Diamond^{s}_{[a,b]}\neg\varphi \\
\Box^{s}_{[a,b]}\varphi & = & \neg\Diamond^{w}_{[a,b]}\neg\varphi \\
\varphi_1 \,\mathcal{U}^{w} \varphi_2 & = & \varphi_1 \,\mathcal{U}^{s} \varphi_2 \vee \Box^{w}\varphi_1 \\
\varphi_1 \,\mathcal{U}^{w}_{[a,b]} \varphi_2 & = & \varphi_1 \,\mathcal{U}^{s}_{[a,b]} \varphi_2 \vee \Box^{w}_{[0,b]}\varphi_1
\end{array}$$

The untimed *eventually* (*always*) comes only in strong (weak) form, as the weak (strong) form of *eventually* (always) is trivially satisfied (falsified) by any trace.

3 STL/PSL Syntax and Semantics

STL/PSL analog extension is built on top of the STL^f temporal logic. It is intended to describe temporal properties on finite-length real-valued and/or Boolean traces. Some issued related to the representation of finite-length real-valued signals are discussed in the analog layer section. One of the main efforts has been done on integrating the logic according to the standard PSL syntax. In this Section we present different STL/PSL constructs, describing their syntax and semantics. We use the notation `x,x1,x2,y,y1,y2` and `phi,phi1,phi2` for formulae from the analog and the temporal layer respectively.

3.1 Analog Layer

The analog layer of STL/PSL allows reasoning about real-valued variables. While STL^f defines analog expressions in an abstract way, STL/PSL provides concrete built-in functions. The choice of functions included in the language has been the result of the discussions with analog designers of ST Microelectronics Italy and their feedback. These functions are at heart of enabling the expression of richer temporal properties, where analog signals can "communicate" between themselves directly, and not only via Boolean abstraction. Some of the operators are memoryless, like the absolute value function, while other operators like `shift` require memory. The analog layer has been designed to be easily extendable to new built-in functions.

Note that the analog functions in STL/PSL manipulate continuous signals that are defined as *ideal mathematical objects* consisting of an uncountable number of pairs $(t, \xi_a[t])$ for all $t \in [0,r)$. Such signals do not admit an *exact finite representation* which can be a problem in implementing and interpreting the STL/PSL analog layer in tools. In fact, the signals generated by numerical simulators usually produce a *finite* collection of sampling points $(t, \xi_a[t])$ with t ranging over some interval $[0,r) \subseteq \mathcal{R}_{\geq 0}$, where $\xi_a[t]$ is stored using floating point representation. Hence, we leave the definitions in the analog layer as general as possible, and delegate to the tools supporting the STL/PSL language to approximate the sampled input signals to their continuous representation. This can be done for example using *piecewise-constant* or *linear* interpolation, which "fill" the missing values between two samples. The latter one is implemented in the STL/PSL Monitor tool, presented in Deliverable 3.2/13 [NM06a].

Absolute value

Syntax: `abs(x)`

`x:` analog expression

`returns:` analog expression

Informal description: `abs(x)` returns the absolute value of `x` as shown in Figure 2.

Definition:

$$\mathtt{abs(x)}[t] = \begin{cases} \mathtt{x}[t] & \text{if } \mathtt{x}[t] \geq 0 \\ -\mathtt{x}[t] & \text{else if } \mathtt{x}[t] < 0 \\ \bot & \text{otherwise} \end{cases}$$

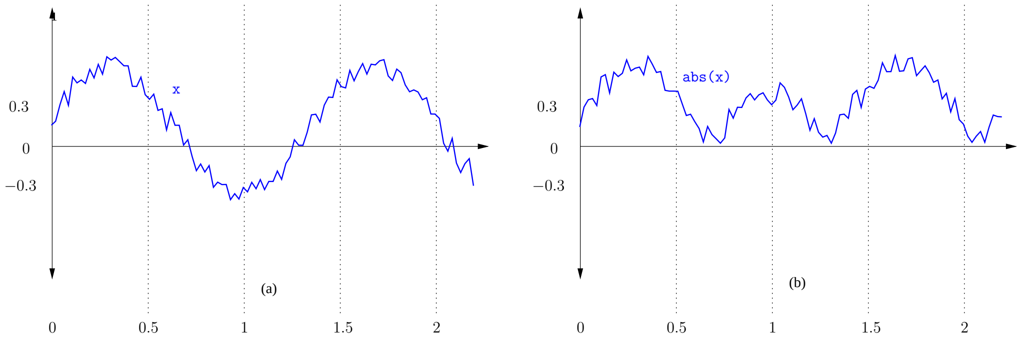


Figure 2: Absolute value: (a) `x` (b) `abs(x)`

Derivative

Syntax: `ddt(x)`

`x:` analog expression

Informal description: `ddt(x)` computes the rate of change of the signal `x` (see Figure 3).

Definition:

$$\mathtt{ddt(x)}[t] = \begin{cases} \frac{d\mathtt{x}[t]}{dt} & \text{if } \mathtt{x}[t] \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

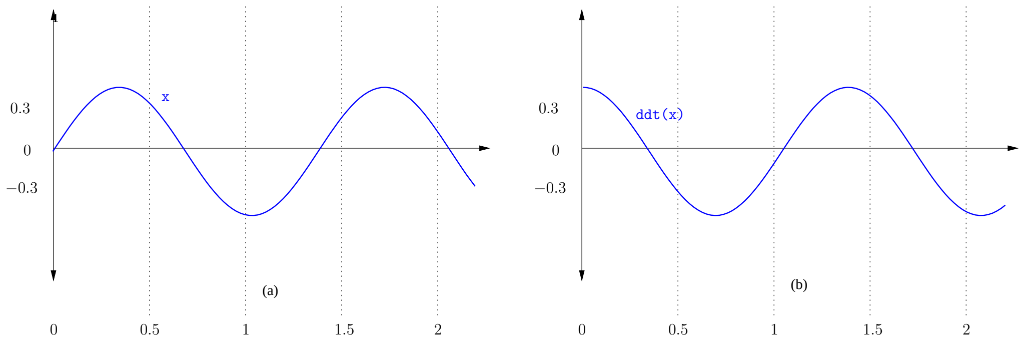


Figure 3: Derivative: (a) `x` (b) `ddt(x)`

Arithmetic Operations

Syntax:

`x1+x2`, `x1-x2`, `x1*x2`, `x1+c`, `x1-c`, `x1*c`

`x1,x2:` analog expressions

`c:` real constant

Informal description: The arithmetic operations are defined in a straightforward fashion as pointwise addition, subtraction and multiplication operations on signal `x` and a constant `c` or another signal `y`. The Figure 4 shows the result of the subtraction of signals `x` and `y`

$$(\mathtt{x1}\star\mathtt{x2})[t] = \begin{cases} \mathtt{x1}[t]\star\mathtt{x2}[t] & \text{if } \mathtt{x1}[t] \neq \bot \text{ and } \mathtt{x2}[t] \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

$$(\mathtt{x}\star\mathtt{c})[t] = \begin{cases} \mathtt{x}[t]\star\mathtt{c} & \text{if } \mathtt{x}[t] \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

where $\star \in \{$`+`, `-`, `*`$\}$.

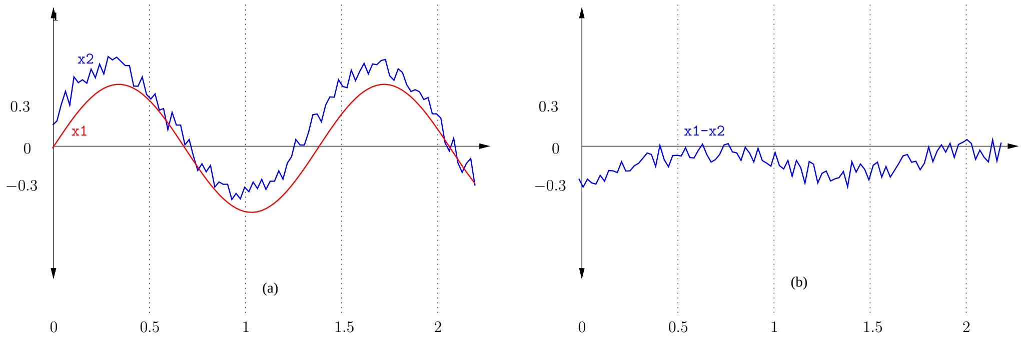


Figure 4: Arithmetic operations: (a) `x1` and `x2` (b) `x1-x2`

Shift

Syntax: `shift(x,c)`

`x`: analog expression `c`: positive real constant

Informal description: The result of this operation is the signal `x` that is shifted by the amount of `c` as shown in Figure 5.

Definition:

$$\texttt{shift(x,c)}[t] = \begin{cases} \texttt{x}[t+\texttt{c}] & \text{if } \texttt{x}[t+\texttt{c}] \neq \perp \\ \perp & \text{otherwise} \end{cases}$$

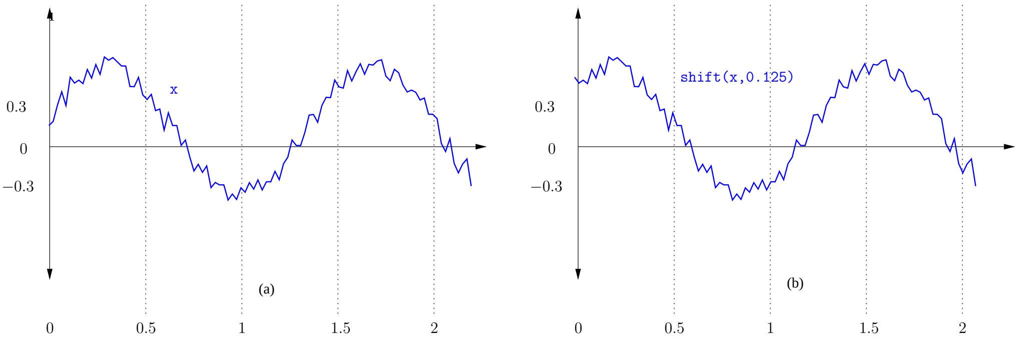


Figure 5: Shift: (a) `x` (b) `shift(x,0.125)`

3.2 Boolean Abstraction

The Boolean abstraction allows mapping continuous signals to Boolean signals. It is an important part of STL/PSL as the temporal properties are interpreted over the Boolean abstractions of the analog inputs rather than on the continuous signals themselves.

The abstraction that partitions the continuous state-space according to the satisfaction of some inequality constraint on the real variables is called *threshold Boolean abstraction*. The Figure 6 (b) is an example of the Boolean mapping of the signal `x` from Figure 6 (a) with respect to a threshold `eps=0.3`.

Although the Boolean abstraction operator is simple, when combined properly with the operators from the analog layer, it becomes much more expressive. As an example, we can express linear constraints of type `5x <= 2y` in STL/PSL by using arithmetic operators and rewriting the constraint as `x*5 - y*2 <= 0`.

Syntax:

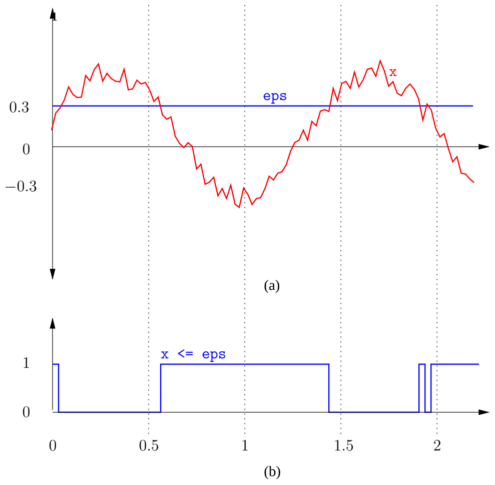


Figure 6: Threshold-based Boolean Abstraction: (a) `x` (b) `x <= 0.3`

`x < eps` — `x <= eps` | `x > eps` | `x >= eps`

`x`: Boolean expression

`eps`: real constant

Informal description: The threshold Boolean abstraction `x <= eps` holds at time `t` iff the value of `x(t)` is smaller or equal to the threshold constant `eps`. We can see an example in Figure 6 where `eps=0.3`.

Semantics:

$$\begin{array}{lcl}
(\xi,t) \models \texttt{x < eps} & \leftrightarrow & \texttt{x}[t] < \texttt{eps} \\
(\xi,t) \models \texttt{x <= eps} & \leftrightarrow & \texttt{x}[t] \leq \texttt{eps} \\
(\xi,t) \models \texttt{x > eps} & \leftrightarrow & \texttt{x}[t] > \texttt{eps} \\
(\xi,t) \models \texttt{x >= eps} & \leftrightarrow & \texttt{x}[t] \geq \texttt{eps}
\end{array}$$

Rewrite rules:

```
x > eps = not (x <= eps)
```

```
x >= eps = not (x < eps)
```

3.3 Temporal Layer

The syntax of the temporal layer of STL/PSL is adapted to the original syntax of PSL, and we use similar syntactic sugaring in order to derive new operators from the basic ones.

Boolean Operators

The Boolean operators in STL/PSL are defined in the usual manner and are interpreted over dense time.

Syntax:

`not phi1` | `phi1 or phi2` | `phi1 and phi2`

| `phi1 -> phi2` | `phi1 <-> phi2` | `phi1 xor phi2`

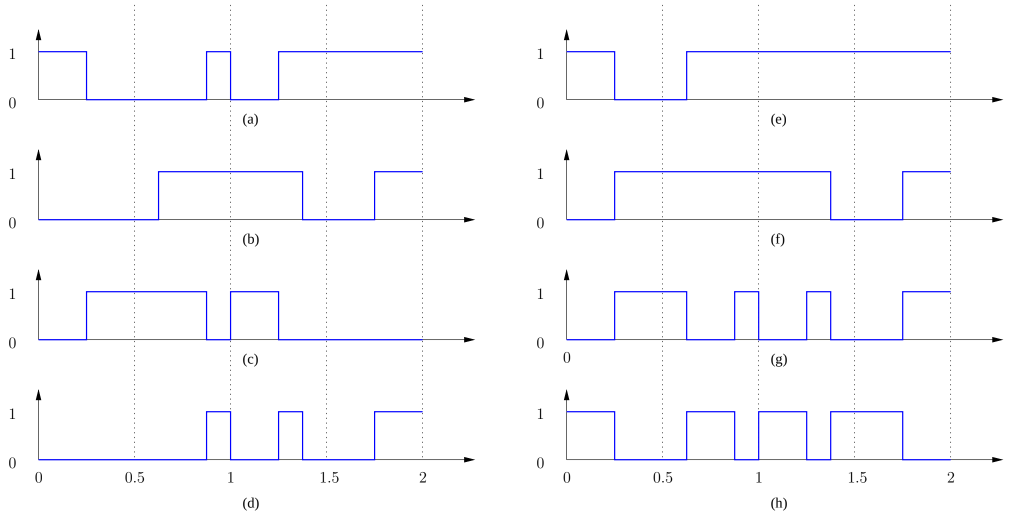
`phi1,phi2:` Boolean expression

Semantics:

$$\begin{array}{lcl}
(\xi,t) \models \texttt{not phi1} & \leftrightarrow & (\xi,t) \not\models \texttt{phi1} \\
(\xi,t) \models \texttt{phi1 or phi2} & \leftrightarrow & (\xi,t) \models \texttt{phi1} \text{ or } (\xi,t) \models \texttt{phi2} \\
(\xi,t) \models \texttt{phi1 and phi2} & \leftrightarrow & (\xi,t) \models \texttt{phi1} \text{ and } (\xi,t) \models \texttt{phi2} \\
(\xi,t) \models \texttt{phi1 -> phi2} & \leftrightarrow & (\xi,t) \models \texttt{phi1} \text{ implies that } (\xi,t) \models \texttt{phi2} \\
(\xi,t) \models \texttt{phi1 <-> phi2} & \leftrightarrow & (\xi,t) \models \texttt{phi1} \text{ iff } (\xi,t) \models \texttt{phi2} \\
(\xi,t) \models \texttt{phi1 xor phi2} & \leftrightarrow & (\xi,t) \models \texttt{phi1} \text{ xor } (\xi,t) \models \texttt{phi2}
\end{array}$$

Rewrite rules:

```
p and q = not (not p or not q)
p -> q = not p or q
p <-> q = (p and q) or (not p and not q)
p xor q = (not p and q) or (p and not q)
```



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 7: Boolean operators: (a) `p` (b) `q` (c) `not p` (d) `p and q` (e) `p or q` (f) `p -> q` (g) `p <-> q` (h) `p xor q`

Untimed Operators

Temporal operators that are untimed (not bounded by an interval) preserve their standard PSL semantics, and are interpreted over real time.

Syntax:

```
always phi
eventually!  phi
phi1 until!  phi2
phi1 until phi2
```

`phi,phi1,phi2:` Boolean expression

Semantics:

$$
\begin{array}{lcl}
(\xi,t) \models \texttt{always phi} & \leftrightarrow & \forall t' \in [t,\infty) \cap \mathbb{T}_\xi \ (\xi,t') \models \texttt{phi} \\
(\xi,t) \models \texttt{eventually! phi} & \leftrightarrow & \exists t' \in [t,\infty) \cap \mathbb{T}_\xi \ (\xi,t') \models \texttt{phi} \\
(\xi,t) \models \texttt{phi1 until! phi2} & \leftrightarrow & \exists t' \in [t,\infty) \cap \mathbb{T}_\xi \text{ st } (\xi,t') \models \texttt{phi2} \text{ and} \\
 & & \forall t'' \in [t,t'] \ (\xi,t) \models \texttt{phi1} \\
(\xi,t) \models \texttt{phi1 until phi2} & \leftrightarrow & (\xi,t) \models \texttt{phi1 until! phi2} \text{ or} \\
 & & (\xi,t) \models \texttt{always phi1}
\end{array}
$$

Rewrite rules:

```
phi1 until phi2 = (phi1 until!  phi2) or always phi1
```

Notes: Due to the “handshake” semantics of the *until* operator in the dense time framework, the PSL operators `until_` and `until` are equivalent, and hence the former is not included in the STL/PSL extension.

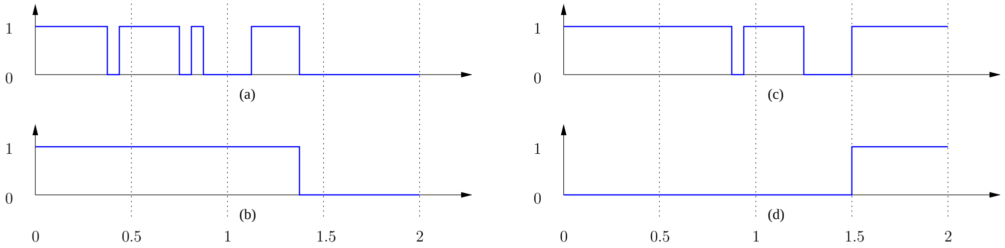


Figure 8: Untimed Eventually and Always: (a) `p` (b) `eventually!p` (c) `q` (d) `always q`

3.4 Timed Operators

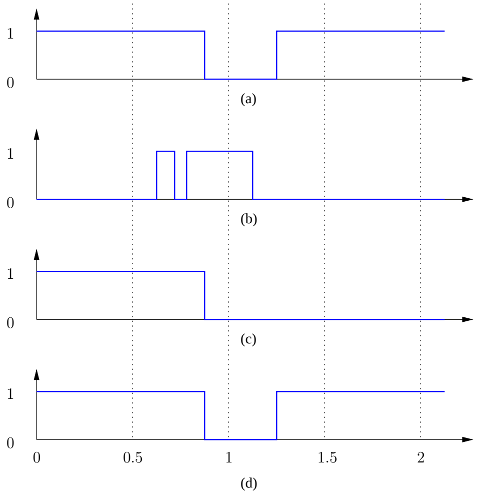


Figure 9: Until and Weak Until: (a) `p` (b) `q` (c) `p until!q` (d) `p until q`

Timed eventually operators

Syntax:

```
eventually![a:b] phi
eventually [a:b] phi
eventually!  [<=b] phi
eventually [<=b] phi
eventually!  [>=a] phi
```

`phi`: Boolean expression

Semantics:

$$
\begin{array}{lll}
(\xi,t) \models \texttt{eventually![a:b] phi} & \leftrightarrow & \exists t' \in [t+a, t+b] \cap \mathbb{T}_\xi \; (\xi,t') \models \texttt{phi} \\
(\xi,t) \models \texttt{eventually[a:b] phi} & \leftrightarrow & \exists t' \in [t+a, t+b] \cap \mathbb{T}_\xi \; (\xi,t') \models \texttt{phi} \\
 & & \text{or } |\xi| \leq t+b \\
(\xi,t) \models \texttt{eventually![<=b] phi} & \leftrightarrow & \exists t' \in [t, t+b] \cap \mathbb{T}_\xi \; (\xi,t') \models \texttt{phi} \\
(\xi,t) \models \texttt{eventually[<=b] phi} & \leftrightarrow & \exists t' \in [t, t+b] \cap \mathbb{T}_\xi \; (\xi,t') \models \texttt{phi} \\
 & & \text{or } |\xi| \leq t+b \\
(\xi,t) \models \texttt{eventually![>=b] phi} & \leftrightarrow & \exists t' \in [t+a, \infty) \cap \mathbb{T}^\xi \; (\xi,t') \models \texttt{phi}
\end{array}
$$

Notes:

Intuitively, `eventually![a:b]` operator corresponds to the dense time extension of the `next_e![a:b]` PSL operator. We adapt the syntax to the standard MITL notation, and to the fact that the *next* operator does not exist in dense time.

The weak version of `eventually![>=a]` does not exist, since as in the untimed case, it trivially evaluates to *true* for all the finite length signals.

Syntax:

```
always![a:b] phi
```

```
always [a:b] phi
```

```
always![<=b] phi
```

```
always [<=b] phi
```

```
always [>=a] phi
```

`phi:` Boolean expression

Semantics:

$$
\begin{array}{lcl}
(\xi,t) \models \texttt{always[a:b] phi} & \leftrightarrow & \forall t' \in [t+a, t+b] \cap \mathbb{T}_\xi \; (\xi,t') \models \texttt{phi} \\
(\xi,t) \models \texttt{always![a:b] phi} & \leftrightarrow & \forall t' \in [t+a, t+b] \cap \mathbb{T}_\xi \; (\xi,t') \models \texttt{phi} \\
 & & \text{and } |\xi| > t+b \\
(\xi,t) \models \texttt{always[<=b] phi} & \leftrightarrow & \forall t' \in [t, t+b] \cap \mathbb{T}_\xi \; (\xi,t') \models \texttt{phi} \\
(\xi,t) \models \texttt{always![<=b] phi} & \leftrightarrow & \forall t' \in [t, t+b] \cap \mathbb{T}_\xi \; (\xi,t') \models \texttt{phi} \\
 & & \text{and } |\xi| > t+b \\
(\xi,t) \models \texttt{always[>=a] phi} & \leftrightarrow & \forall t' \in [t+a, \infty) \cap \mathbb{T}_\xi \; (\xi,t') \models \texttt{phi}
\end{array}
$$

Notes:

Intuitively, `always[a:b]` operator corresponds to the dense time extension of the `next_a[a:b]` PSL operator. We adapt the syntax to the standard MITL notation, and to the fact that the *next* operator does not exist in the dense time.

The strong version of `always [>=a]` does not exist, since as in the untimed case, it trivially evaluates to $false$ for all the finite length signals.

Timed until operators

Syntax:

```
phi1 until![a:b] phi2
```

```
phi1 until [a:b] phi2
```

```
phi1 until![<=b] phi2
```

```
phi1 until [<=b] phi2
```

```
phi1 until [>=a] phi2
```

```
phi1 until [>=b] phi2
```

`phi1,phi2:` Boolean expression

Semantics:

$$\begin{aligned}
(\xi,t) \models \texttt{phi1 until![a:b] phi2} &\leftrightarrow \exists t' \in [t+a, t+b] \cap \mathbb{T}_{\xi} \text{ st } (\xi,t') \models \texttt{phi2} \text{ and} \\
&\quad \forall t'' \in [t,t'] \; (\xi,t) \models \texttt{phi1} \\
(\xi,t) \models \texttt{phi1 until[a:b] phi2} &\leftrightarrow (\xi,t) \models \texttt{phi1 until![a:b] phi2} \text{ or} \\
&\quad (\xi,t) \models \texttt{always[<=b] phi1} \\
(\xi,t) \models \texttt{phi1 until![<=b] phi2} &\leftrightarrow \exists t' \in [t, t+b] \cap \mathbb{T}_{\xi} \text{ st } (\xi,t') \models \texttt{phi2} \text{ and} \\
&\quad \forall t'' \in [t,t'] \; (\xi,t) \models \texttt{phi1} \\
(\xi,t) \models \texttt{phi1 until[<=b] phi2} &\leftrightarrow (\xi,t) \models \texttt{phi1 until![<=b] phi2} \text{ or} \\
&\quad (\xi,t) \models \texttt{always[<=b] phi} \\
(\xi,t) \models \texttt{phi1 until![>=a] phi2} &\leftrightarrow \exists t' \in [t+a, \infty) \cap \mathbb{T}_{\xi} \text{ st } (\xi,t') \models \texttt{phi2} \text{ and} \\
&\quad \forall t'' \in [t,t'] \; (\xi,t) \models \texttt{phi1} \\
(\xi,t) \models \texttt{phi1 until[>=a] phi2} &\leftrightarrow (\xi,t) \models \texttt{phi1 until![>=a] phi2} \text{ or} \\
&\quad (\xi,t) \models \texttt{always phi1}
\end{aligned}$$

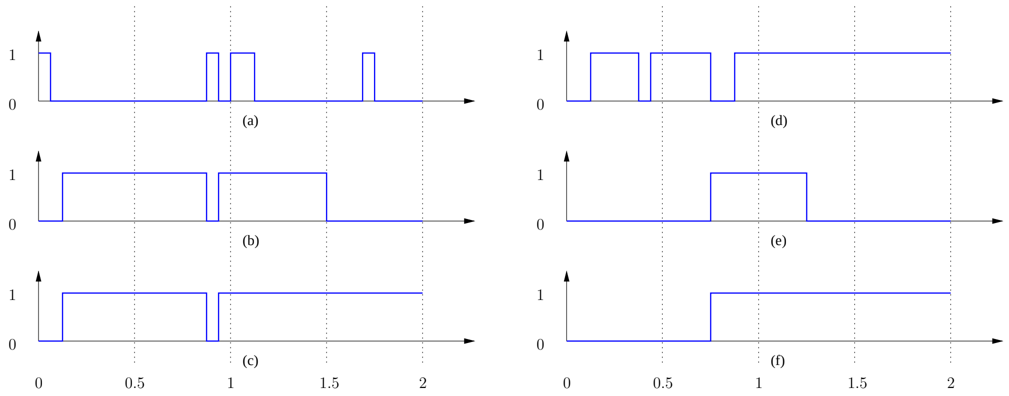


Figure 10: Bounded eventually and always operators: (a) `p` (b) `eventually! [0.25:0.75] p` (c) `eventually [0.25:0.75] p` (d) `q` (e) `always![0.25:0.75] q` (f) `always [0.25:0.75] q`

3.5 Distance-based properties

A large part of analog design is based on comparing waveforms (signals) with some reference signal that specify a desired behavior. These notions are formalized using a distance function (metric) which quantifies numerically the resemblance of two signals. Mathematically speaking, a metric space is a pair (X,d) such that X is the domain and $d : X \times X \to \mathbb{R}_+$ is a function satisfying: $d(x,x) = 0$; $d(x,y) = d(y,x)$ and $d(x,y) + d(y,z) \geq d(x,z)$. There are many ways to define distance function on waveforms, by taking the maximum of the pointwise distance at every time t, summing/integrating over the pointwise distance, etc. Once such a

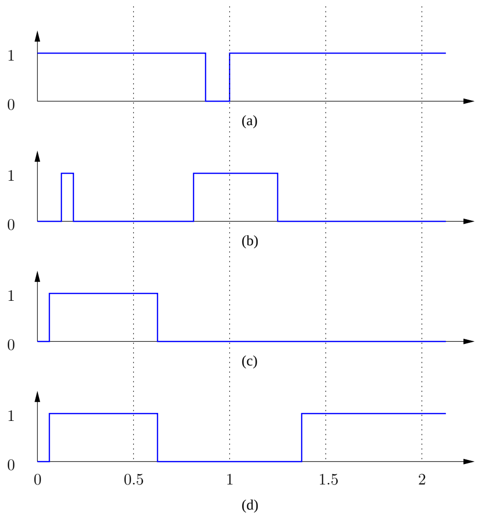


Figure 11: Bounded until operators: (a) `p` (b) `p` (c) `p until! [0.25:0.75] q` (d) `p until [0.25:0.75] q`

distance d is defined, it can be used to define distance-based properties of the form $d(\xi,\xi') < c$ for some positive constant c.

Below we define several properties of this type based on the maximal pointwise distance [FGP06] and on a new variant of it which tolerated delays and is particularly suited for mixed signals [KC06a, KC06b]. The distance-based properties represent an effort of enriching STL/PSL with metrical properties. However, we have found equivalent STL/PSL formulae that can represent such distance operators.

3.6 Threshold distance

Syntax: `distance (x,y,eps)`

`x,y`: analog expression
`eps`: threshold

Informal description: The threshold distance holds at time `t` iff the values of `x(t)` and `y(t)` do not deviate from each other more than the threshold `eps`. This distance is a pointwise operator that allows to define an acceptable bound on how much two signals can differ one from the other.

The threshold distance is particularly useful in order to compare the output of a simulation to a reference signal. The typical property using the threshold distance would be `always distance (x,y,eps)`. In Figure 12 (a) and

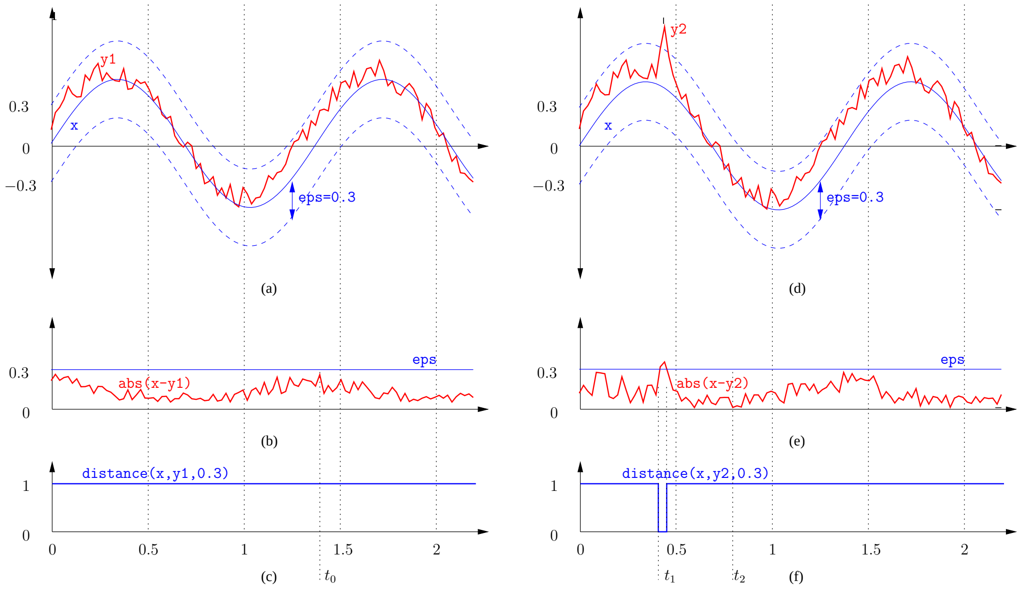


Figure 12: Threshold distance: (a) x and y1 (b) abs(x-y1) (c) distance (x,y1,0.3) (d) x and y2 (e) abs(x-y2) (f) distance (x,y2,0.3)

(d), we can see two signals y1 and y2 compared to the reference signal x. The maximum relative deviation from each other is denoted by the constant eps=0.3. We can see from the Figure 12 (c) that distance(x,y1,0.3) is satisfied at all time points. On the other hand, the property distance(x,y2,0.3) fails at t_1 (Figure 12 (f), because the signal x2 has a peak (of short duration) whose value with respect to the value of the reference signal y at the same time is greater than 0.3.

The threshold distance returns a Boolean value at different time points representing its satisfaction at that time, some useful qualitative measures can be extracted from the pointwise comparison of signals x and y. |x(t)-y(t)| is a value that compared to the threshold eps gives the actual level of similarity between x and y at time t as we can see from Figure 12 (b) and (e). For instance |x(t)-y(t)|=0 represents perfect equivalence of x and y at time t (this is the case for the signals x and y2 at time t_2, as shown in Figure12 (e)), while any value |x(t)-y(t)| smaller than eps but close to it indicates that the threshold distance property between x and y still holds at time t but is close to the failure. This is the case for the signals x and y1 at time t_0 in Figure 12 (b), indicating that the distance property holds, but is not robust with respect to the given threshold. As we can see, taking the maximum absolute difference between the two signals, we can infer the degree of robustness between two signals with respect to their threshold distance.

Rewrite rules: distance (x,y,eps) = abs(x-y) <= eps

Notes: The pointwise threshold distance compares only continuous signals. The Boolean pointwise distance operator corresponds to <-> operator.

3.7 Threshold-delay distance

Syntax: `distance (x,y,eps,T1,T2)` | `distance (p,q,T1,T2)`

`x,y`: analog expression
`p,q`: Boolean expression
`eps`: threshold
`T,T'`: large and small time windows

Informal Description: The threshold-delay distance extends the threshold distance operator by adding a delay tolerance on the values of signals `x` and `y`. The delays tolerated are defined by the parameters `T1` and `T2` representing the lengths of two time windows such that `T2 < T1`. The threshold-delay distance is similar to the threshold distance operator in that it requires that the absolute difference `|x(t)-y(t)|` between the two signals be smaller than the threshold `eps`. However, it allows this condition to become false, but for small periods of time, not greater than `T2`. Moreover, such episodes should not be too frequent, ie. any period of time where `|x(t)-y(t)|>eps` has to be followed by another period lasting at least for `T1-T2` time, and where `|x(t)-y(t)|<=eps` remains continuously true.

In other words, the threshold-delay distance `distance(x,y,eps,T1,T2)` holds at time `t` iff within the time window `I=[t,t+T1]`, there is a smaller time window `I'=[t',t'+T1-T2]` such that `I'` ⊂ I where the absolute difference of `x` and `y` continuously remains below `eps`.

As examples, consider the property `distance(x,y,0.3,0.375,0.125)` and different cases shown in Figures 13 and 14. In Figure 13, the signal `y1` has a peak at time t_0 such that `abs(x,y1)` is greater than `0.3`. Given that this period lasts for less than `0.125` time units, the threshold-delay distance holds at t_0. Note that the pointwise threshold distance would fail at time t_0 for the same inputs. The Figure 13 (d) shows the similar case, but now we have two episodes where the threshold distance between `x` and `y2` is greater than `0.3`, that is during the intervals $[t_1,t_2)$ and $[t_3,t_4)$. Since both episodes are smaller than `T2=0.125` and between t_2 and t_3 their pointwise distance remains continuously smaller than `0.3` for more than `T1-T2=0.25` time units, the threshold-delay distance also holds at all time points.

Now consider the Figure 14 (a). At time t_0, the absolute difference between `x` and `y1` becomes greater than `0.3`, and remains continuously above the threshold for more than 0.125 time units. Hence, the threshold-delay distance property is false at t_0. Finally, in Figure 14 (d), there are two episodes where `abs(x-y1)` is above `0.3`, starting at t_1 and t_3 respectively. Both episodes have a duration smaller than `T2=0.125`, but since the duration in between them where `abs(x-y1)<=0.3` holds is smaller than `T1-T2=0.25`, the property fails at t_1 too.

The threshold-delay distance operator can also be used to compare two Boolean signals. In this case, the threshold is not specified, as two Boolean signals can either have the same or the opposite values at a given time point. Hence, two signals `p` and `q` are similar with respect to their threshold-delay distance at time `t`, iff within the time window `I=[t,t+T1]`, there is a smaller time window `I'=[t',t'+T1-T2]` such that `I'` ⊂ I and `p <-> q` throughout `I'`. An example of a Boolean threshold-distance operator applied to signals `p` and `q` is shown in Figure 15.

Rewrite rules: The threshold-delay distance can be fully expressed using the basic operators from the analog and temporal layer.

```
distance(x,y,eps,T1,T2)   =  (abs(x-y) > eps) ->
                             eventually!  [0:T1]
                             always [0:T1-T2]
                             (abs(x-y) <= eps)
distance(p,q,T1,T2)       =  (p xor q) ->
                             eventually!  [0:T1]
                             always [0:T1-T2]
                             (p <-> q)
```

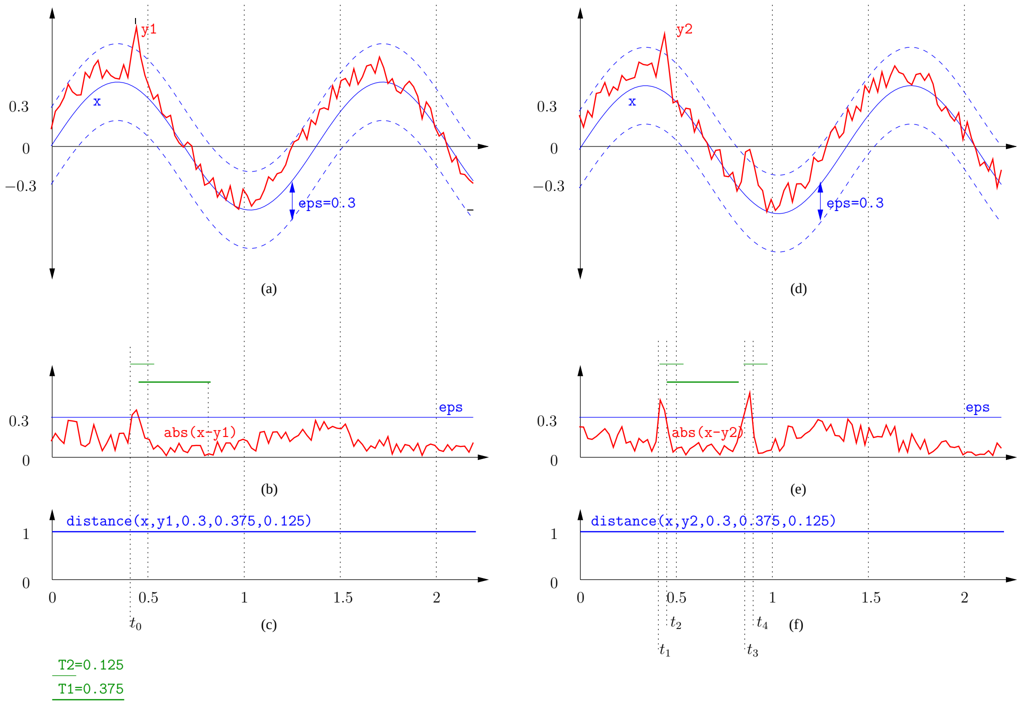


Figure 13: Analog Threshold-Delay Distance: Property Holds (a) `x` and `y1` (b) `|x-y1|` (c) `distance(x,y1,0.3,0.125,0.375)` (d) `x` and `y2` (e) `|x-y2|` (f) `distance(x,y2,0.3,0.125,0.375)`

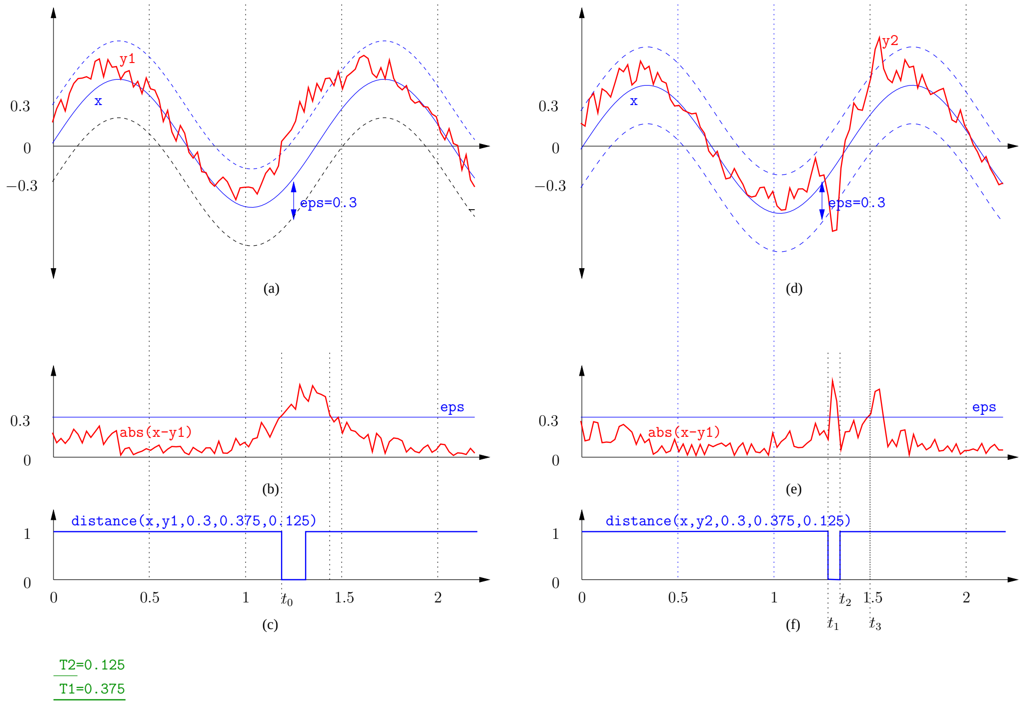


Figure 14: Analog Threshold-Delay Distance: Property Fails (a) `x` and `y1` (b) `|x-y1|` (c) `distance(x,y1,0.3,0.125,0.375)` (d) `x` and `y2` (e) `|x-y2|` (f) `distance(x,y2,0.3,0.125,0.375)`

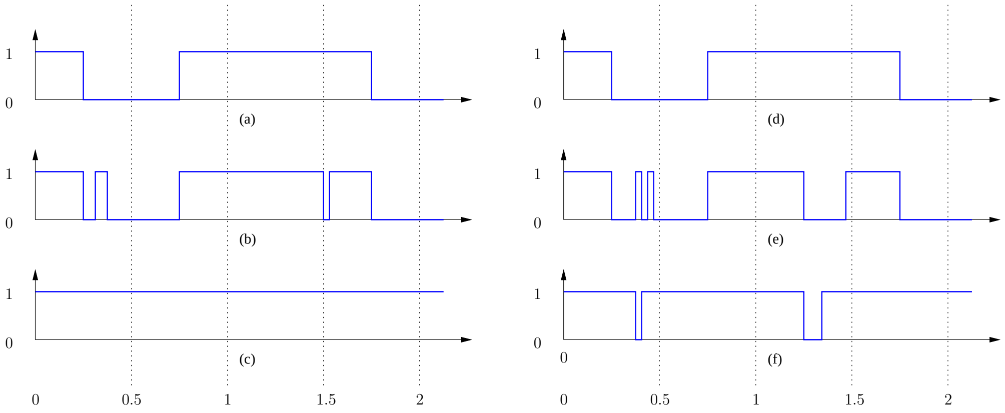


Figure 15: Boolean Threshold-Delay Distance: (a) `p` (b) `q1` (c) `distance(p,q1,0.125,0.375)` (d) `p` (e) `q1` (f) `distance(p,q1,0.125,0.375)`

4 Conclusion

We have presented in this document the final proposal for an analog extension of the PSL language. The decisions on features that have been embedded in STL/PSL were driven by both theoretical constraints an practical feedback from the analog designers and have been discussed in Section 1. The analog layer allows to express richer temporal properties and to relate continuous signals directly. The Boolean abstraction is used in STL/PSL as a connection between analog and temporal layers. The temporal layer has been extended in order to treat dense time. Finally, a study on metric distances has resulted in new operators allowing to compare simulation outputs to a reference signal in a robust manner. As a pioneering work in the domain, we believe that STL/PSL presents a foundation for further exploration of property validation of analog systems.

5 References

[ACM02] E. Asarin, P. Caspi and O. Maler, Timed Regular Expressions, *The Journal of the ACM* **49**, 172–206, 2002.

[AFH96] R. Alur, T. Feder, and T.A. Henzinger. The Benefits of Relaxing Punctuality. *Journal of the ACM*, 43(1):116–146, 1996.

[ASS+05] B. d'Angelo, S. Sankaranarayanan, C. Sanchez, W. Robinson, B. Finkbeiner, H. B. Sipma, S. Mehrotra, Z. Manna, LOLA: Runtime Monitoring of Synchronous Systems In *Proc. TIME'05*, pages 166–174. IEEE Computer Society Press, 2005.

[FGP06] G. Fainekos, A. Girard and G. Pappas, Temporal Logic Verification Using Simulation In *Proc. FORMATS'06*, pages 171–186. LNCS 4202, Springer, 2006.

[HFE04] J. Havlicek, D. Fisman and C. Eisner, Basic results on the semantics of Accellera PSL 1.1 foundation language, *Technical Report 2004.02*, Accelera, 2004.

[KC06a] C. Konsentini and P. Caspi, Sampling and Voting in Hybrid Computing Systems In *Proc. HSCC'06*.

[KC06b] C. Konsentini and P. Caspi, Approximation, Sampling and Voting in Hybrid Computing Systems In *TR-2005-19*, Verimag Technical Report.

[KCV04] A. Kumari B. Cohen and S. Venkataramanan, *Using PSL/Sugar for Formal and Dynamic Verification*, VhdlCohen Publishing, 2004.

[MN04] O. Maler and D. Nickovic, Monitoring Temporal Properties of Continuous Signals, *FORMATS/FTRTFT'04*, 152-166, LNCS 3253, 2004.

[MNP06] O. Maler, D. Nickovic and A. Pnueli, From MITL to Timed Automata In *Proc. FORMATS'06*, pages 274–289. LNCS 4202, Springer, 2006.

[M05] O. Maler, *Extending PSL for Analog Circuits*, PROSYD Deliverable D1.3/1, 2005.

[NM06a] D. Nickovic, O. Maler, *Manual for Property-based automatic generation of simulation monitors for digital, timed, and analog designs*, PROSYD Deliverable D3.2/13, 2006.

[NM06b] D. Nickovic, O. Maler, *Analog Case Study*, PROSYD Deliverable D3.4/2, 2006.

[Y97] S. Yovine, Kronos: A Verification Tool for Real-time Systems, *International Journal of Software Tools for Technology Transfer* **1**, 123–133, 1997.

A Production Rules for STL/PSL

In this Appendix, we present the subset of STL/PSL which represents the extensions with respect to the original PSL and that is ready for embedding with other PSL layers not discussed in this document. The production rules of STL/PSL are expressed in the BNF form.

```
\* Analog Layer *\
```

Analog_Expression :==
Analog_Variable
| **(** Analog_Expression **)**
| **abs (** Analog_Expression **)**
| **ddt (** Analog_Expression **)**
| **shift (** Analog_Expression , REAL **)**
| Analog_Expression **-** Analog_Expression
| Analog_Expression **+** Analog_Expression
| Analog_Expression ***** Analog_Expression
| Analog_Expression **-** REAL
| Analog_Expression **+** REAL
| Analog_Expression ***** REAL

```
\* Boolean Abstraction *\
```

Boolean_Expression :==
Boolean_Variable
| Threshold_Boolean_Abstraction

Threshold_Boolean_Abstraction :==
Analog_Expression <**=** REAL
| Analog_Expression > REAL
| Analog_Expression < REAL
| Analog_Expression >**=** REAL

```
\* Temporal Layer *\
```

Stl_Psl_Property :==
Boolean_Expression
| Distance
| **(** Stl_Psl_Property **)**
| **not** Stl_Psl_Property
| Stl_Psl_Property **or** Stl_Psl_Property
| Stl_Psl_Property **and** Stl_Psl_Property
| Stl_Psl_Property **-**> Stl_Psl_Property
| Stl_Psl_Property <**-**> Stl_Psl_Property
| Stl_Psl_Property **xor** Stl_Psl_Property
| **eventually!** Stl_Psl_Property
| **always** Stl_Psl_Property
| Stl_Psl_Property **until!** Stl_Psl_Property
| Stl_Psl_Property **until** Stl_Psl_Property
| **eventually! [** REAL **:** REAL **]** Stl_Psl_Property
| **eventually [** REAL **:** REAL **]** Stl_Psl_Property
| **eventually! [**<**=** REAL **]** Stl_Psl_Property
| **eventually [**<**=** REAL **]** Stl_Psl_Property
| **eventually! [**>**=** REAL **]** Stl_Psl_Property
| **always! [** REAL **:** REAL **]** Stl_Psl_Property
| **always [** REAL **:** REAL **]** Stl_Psl_Property
| **always! [**<**=** REAL **]** Stl_Psl_Property
| **always [**<**=** REAL **]** Stl_Psl_Property
| **always [**>**=** REAL **]** Stl_Psl_Property
| Stl_Psl_Property **until! [** REAL **:** REAL **]** Stl_Psl_Property
| Stl_Psl_Property **until [** REAL **:** REAL **]** Stl_Psl_Property
| Stl_Psl_Property **until! [**<**=** REAL **]** Stl_Psl_Property
| Stl_Psl_Property **until [**<**=** REAL **]** Stl_Psl_Property
| Stl_Psl_Property **until! [**>**=** REAL **]** Stl_Psl_Property
| Stl_Psl_Property **until [**>**=** REAL **]** Stl_Psl_Property

```
\* Distance-based Operators (part of the Temporal Layer) *\
```

Distance :==
distance (Analog_Expression **,** Analog_Expression **,** REAL **)**
| **distance (** Analog_Expression **,** Analog_Expression **,**
REAL **,** REAL **,** REAL **)**
| **distance (** Boolean_Expression **,** Boolean_Expression **,**
REAL **,** REAL **)**