STL/PSL Language

Oded Maler and Dejan Nickovic

Verimag

1 Introduction

This document describes the STL/PSL specification language. We first introduce *metric interval temporal logic* MITL (see section 2), a dense-time temporal logic that forms the basis for STL/PSL and show how it is extended to *signal temporal logic* STL in section 3. Finally, the specification language STL/PSL is presented in 4 as syntactic sugar of STL actually used within the AMT tool.

2 MITL: a Real-time Temporal Logic

The temporal logic MITL (metric interval temporal logic) is one of the most popular real-time extensions of linear-time temporal logic LTL. We use Boolean signals as the semantic domain. A Boolean signal w is a partial function $w : \mathbb{T} \to \mathbb{R}^n$ where the time domain $\mathbb{T}$ is the set $\mathbb{R}_{\geq 0}$. The principal modality of MITL is the *timed until* $\mathcal{U}_I$ where I is some non-punctual interval with integer or rational endpoints. A formula $p\mathcal{U}_{[a,b]}p$ is satisfied by a signal 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'.

2.1 Syntax and Semantics

We consider the MITL logic with both *future* and *past* operators. The syntax of MITL is defined by the grammar

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

where p belongs to a set $P = \{p_1, \ldots, p_n\}$ of propositions and I is an interval of the form $[a,b]$, $[a,b)$, $(a,b]$, (a,b), $[a,\infty)$ or (a,∞) where $0 \leq a < b$ are rational numbers. As in LTL the basic MITL operators can be used to derive other standard Boolean and temporal operators, in particular the time-constrained *eventually*, *once*, *always*, and *historically* operators:

$$\begin{array}{ll} \Diamond_I\, \varphi = \mathrm{T}\ \mathcal{U}_I\ \varphi & \diamondminus_I\, \varphi = \mathrm{T}\ \mathcal{S}_I\ \varphi \\ \Box_I\, \varphi = \neg \Diamond_I \neg\varphi & \boxminus_I\, \varphi = \neg \diamondminus_I \neg\varphi \end{array}$$

The semantics of an MITL formula φ with respect to an n-dimensional Boolean signal w is described via the satisfiability relation $(w,t) \models \varphi$, indicating that the signal w satisfies φ at time t, according to following recursive definition.

$$
\begin{array}{lcl}
(w,t) \models p & \leftrightarrow & p[t]=1 \\
(w,t) \models \neg\varphi & \leftrightarrow & (\xi,t) \not\models \varphi \\
(w,t) \models \varphi_1 \vee \varphi_2 & \leftrightarrow & (w,t) \models \varphi_1 \text{ or } (w,t) \models \varphi_2 \\
(w,t) \models \varphi_1 \ \mathcal{U}_I \ \varphi_2 & \leftrightarrow & \exists\, t' \in t \oplus I \ (w,t') \models \varphi_2 \text{ and} \\
 & & \forall\, t'' \in (t,t') \ (w,t'') \models \varphi_1 \\
(w,t) \models \varphi_1 \ \mathcal{S}_I \ \varphi_2 & \leftrightarrow & \exists\, t' \in t \ominus I \ (w,t') \models \varphi_2 \text{ and} \\
 & & \forall\, t'' \in (t',t) \ (w,t'') \models \varphi_1
\end{array}
\qquad (1)
$$

A formula φ is satisfied by w if $(w,0) \models \varphi$. Recall that the satisfaction relation can be viewed as characteristic function χ^φ mapping signals over $\mathbb{B}^n$ into Boolean signals, such that $u = \chi^\varphi(w)$ meaning that for every $t \geq 0$, $u[t] = 1$ if and only if $(w,t) \models \varphi$. The definitions of $\mathcal{U}_I$ and $\mathcal{S}_I$ are *strict*, meaning that φ_1 need not hold at t and neither at the moment t' when φ_2 becomes true. Note also that when I is left-open with a bound a, the truth of φ_2 at $t+a$ does not count for satisfaction.

Expressing Events MITL does not provide constructs that allow to reason explicitly about *instantaneous events* which can be viewed as taking place in singular intervals of zero duration. A natural way to introduce them is to consider the instants when a signal changes its value. To this end we propose two unary operators, *rise* $\uparrow$ and *fall* $\downarrow$, which hold at the rising and falling edges of a Boolean signal, respectively. However, since we allow singular points to be equal to their left neighborhood, $\uparrow p$ may hold at t even if $p[t] = 0$ as illustrated in Figure 1. Intuitively, $\uparrow \varphi$ holds at t if φ is false at t and true in a right neighborhood of t, or if φ is true at t and false in a left neighborhood of t. These operators can be expressed in MITL if we allow *both* future and past operators, as follows:

$$
\begin{array}{l}
\uparrow \varphi = (\varphi \wedge (\neg\varphi \ \mathcal{S} \ \mathrm{T})) \ \vee \ (\neg\varphi \wedge (\varphi \ \mathcal{U} \ \mathrm{T})) \\
\downarrow \varphi = (\neg\varphi \wedge (\varphi \ \mathcal{S} \ \mathrm{T})) \ \wedge \ (\varphi \wedge (\neg\varphi \ \mathcal{U} \ \mathrm{T}))
\end{array}
$$

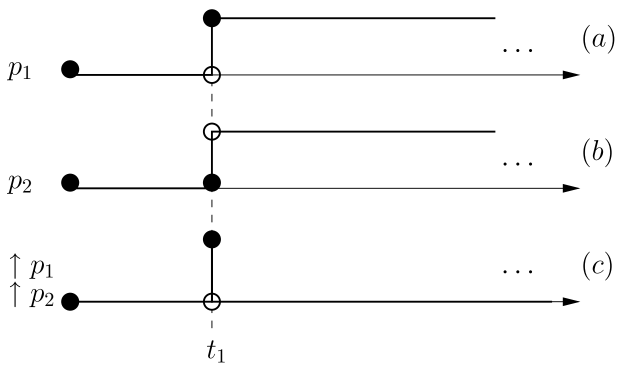

Fig. 1. Two signals p_1 and p_2 that differ at time t where both $\uparrow p_1$ and $\uparrow p_2$ hold.

2.2 Interpretation of MITL over Incomplete Behaviors

The problems related to finitary interpretation of LTL are inherited by dense-time adaptations of temporal logic. We adopt the PSL approach of providing *strong* ($\mathcal{U}_I^s$) and *weak* ($\mathcal{U}_I^w$) versions of the $\mathcal{U}_I$ operator, defined as follows:

$$
\begin{array}{lll}
(w,t) \models \varphi_1\ \mathcal{U}_I^s\ \varphi_2 & \leftrightarrow & \exists\, t' \in t \oplus I \text{ st } (t' < |w| \text{ and } (\xi, t') \models \varphi_2) \text{ and} \\
 & & \forall\, t'' \in (t,t')\ (w,t'') \models \varphi_1 \\
(w,t) \models \varphi_1\ \mathcal{U}_I^w\ \varphi_2 & \leftrightarrow & \exists\, t' \in t \oplus I \text{ st } (t' \geq |w| \text{ or } (\xi, t') \models \varphi_2) \text{ and} \\
 & & \forall\, t'' \in (t,t')\ (w,t'') \models \varphi_1
\end{array}
$$

This definition allows to derive other standard future timed operators *eventually* and *always* in the usual fashion

$$
\begin{array}{ll}
\Diamond_I^s \varphi = \mathrm{T}\ \mathcal{U}_I^s\ \varphi & \Box_I^w \varphi = \neg \Diamond_I^s \neg\varphi \\
\Diamond_I^w \varphi = \mathrm{T}\ \mathcal{U}_I^w\ \varphi & \Box_I^s \varphi = \neg \Diamond_I^w \neg\varphi
\end{array}
$$

3 Signal Temporal Logic STL

In this section we show how MITL is extended toward real-valued (continuous, analog) signals, that is, functions from $\mathbb{R}_{\geq 0}$ to $\mathbb{R}^n$. Such signals form a much richer class of objects and the first issue to be resolved is to define the class of properties that we use. Our choice is to use properties whose checking can be transformed into checking MITL against a "Booleanization" of the signal via finitely many predicates. Once this is defined, all that remains is to handle technical problems related to the (sampled) representation of such signals inside the computer.

3.1 Signal Booleanization and the Logic STL

We consider signals of the form $\xi : \mathbb{T} \to X$ over state-space $X \subseteq \mathbb{R}^n$. A predicate over X is a function $\mu : X \to \mathbb{B}$ which can be syntactically expressed using arithmetic functions and inequalities over the state variable, for example, $x < 5$ or $|x^2 - y^2| \leq 1$. We consider a finite set of such predicates such that by applying them pointwise we obtain Boolean signals describing the evolution over time of the truth values of these predicates with respect to w.

Definition 1 (Booleanization). *Let $\xi : \mathbb{T} \to X$ be a real-valued signal and let $M = \{\mu_1, \ldots, \mu_m\}$ be be a set of predicates of the form $\mu_i : X \to \mathbb{B}$. The M-Booleanization of ξ, denoted by $M(\xi)$, is the signal $w : \mathbb{T} \to \mathbb{B}^m$ satisfying for every i and for every t*

$$w_i[t] = \mu_i(x_1, \ldots, x_n).$$

Events such as *rising* and *falling* in the Boolean signal correspond to some *qualitative* changes in the real-valued signal, for example threshold crossing of some continuous variable.

We know define the *signal temporal logic* STL as an extension of MITL that can express properties that depend on the Booleanization of the signal. That is, we are concerned with properties such that if two signals ξ and ξ' satisfy $M(\xi) = M(\xi')$ then for every formula φ, $\xi \models \varphi$ iff $\xi' \models \varphi$.

The syntax of STL is thus parameterized by a set of real-valued function symbols $f_1, \ldots f_k$. A term of STL is either a rational constant c, a real-valued variable x or a function $f(x_1, \ldots x_n)$. A predicate of STL is an expression of the form $E \sim c$ where E is a term, and $\sim \in \{<, \leq, =, \geq, >\}$. The whole syntax is very much like MITL where predicates have the same role as atomic propositions:

$$\varphi := p \mid E \sim c \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \; \mathcal{U}_I \; \varphi_2 \mid \varphi_1 \; \mathcal{S}_I \; \varphi_2$$

where p belongs to a set $P = \{p_1, \ldots, p_n\}$ of propositions, E is a term, $\sim \in \{<, \leq, =, \geq, >\}$, c is a constant and I is an interval of the form $[a, b],\ [a, b),\ (a, b],\ (a, b),\ [a, \infty)$ or (a, ∞) where $0 \leq a < b$ are rational numbers.

Example An example of a property that can be expressed in STL is a mixed signal stabilization property that has the following requirements:

- The absolute value of a continuous signal x is always less than 6
- When the (Boolean) trigger rises, within 600 time units $|x|$ has to drop below 1 and stay like that for at least 300 time units

This property is illustrated in Figure 2 and expressed in STL as:

$$\Box(|x| < 6 \wedge (\uparrow \mathit{trigger} \rightarrow \Diamond_{[0,600]} \Box_{[0,300]}(|x| < 1)))$$

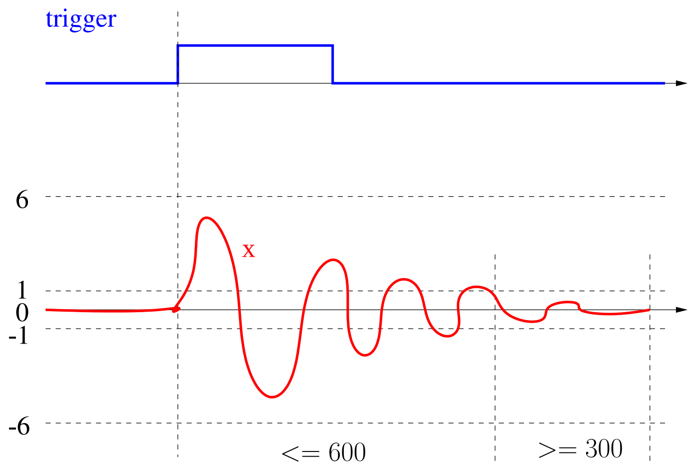

Fig. 2. Mixed signal stabilization property

The monitoring of STL can be easily reduced to Booleanization and monitoring against the MITL-skeleton of the formula.

4 STL/PSL Language

STL/PSL language provides syntactic sugar to STL and is inspired by the PSL language with additional constructs intended to simplify the process of property specification. Each top-level STL/PSL property is declared as an *assertion*, and a number of assertions can be grouped into a single logical unit in order to monitor them simultaneously. We also add a definition directive which allows the user to declare a formula and give it a name, and then refer to it as a variable within the assertions. The syntax of STL/PSL is defined with the following production rules

```
varphi :==
  vprop NAME {
      { define_directive } { assert_directive }
      }

define_directive :==
  define b:NAME := varphi
  | define a:NAME := phi

assert_directive :==
  NAME assert : varphi
```

where `varphi` corresponds to a temporal property and `phi` to an analog operation. The set of operators that are included in STL/PSL is summarized in Tables 1 and 2[1].

[1] STL/PSL supports all the combinations of strength for future temporal operators, with strong operators having the `!`-suffix. Hence, for example both `always[5:10]` and `always![5:10]` are allowed, corresponding to $\Box^{w}_{[5,10]}$ and $\Box^{s}_{[5,10]}$ operators, respectively. In Table 1 we show only (more common) strong versions of $\mathcal{U}_I$ and $\Diamond_I$ and the weak version of $\Box_I$

STL/PSL	STL	Description
`a:x`	x	Analog variable

`phi1 - phi2`
`phi1 + phi2`
`phi1 * phi2`
`phi1 - c`
`phi1 + c`
`phi1 * c`
`abs(phi)`
	$\phi_1 - \phi_2$
	$\phi_1 + \phi_2$
	$\phi_1 * \phi_2$
	$\phi_1 - c$
	$\phi_1 + c$
	$\phi_1 * c$
	$\lvert\phi\rvert$
		Analog operators

`phi <= c`
`phi < c`
`phi >= c`
`phi > c`
`phi == c`
	$\phi \leq c$
	$\phi < c$
	$\phi \geq c$
	$\phi > c$
	$\phi = c$
		Predicates

`b:p`	p	Boolean proposition

`not varphi1`
`varphi1 or varphi2`
`varphi1 and varphi2`
`varphi1 -> varphi2`
`varphi1 <-> varphi2`
`varphi1 xor varphi2`
	$\neg\varphi$
	$\varphi_1 \vee \varphi_2$
	$\varphi_1 \wedge \varphi_2$
	$\varphi_1 \rightarrow \varphi_2$
	$\varphi_1 \leftrightarrow \varphi_2$
	$\varphi_1 \neq \varphi_2$
		Boolean operators

`eventually! varphi`
`eventually!(a:b) varphi`
`eventually![a:b) varphi`
`eventually!(a:b] varphi`
`eventually![a:b] varphi`
`eventually![>b] varphi`
`eventually![>=b] varphi`
`always varphi`
`always(a:b) varphi`
`always[a:b) varphi`
`always(a:b] varphi`
`always[a:b] varphi`
`always[>b] varphi`
`always[>=b] varphi`
`varphi1 until! varphi2`
`varphi1 until!(a:b) varphi2`
`varphi1 until![a:b) varphi2`
`varphi1 until!(a:b] varphi2`
`varphi1 until![a:b] varphi2`
`varphi1 until![>b] varphi2`
`varphi1 until![>=b] varphi2`
	$\Diamond^{s} \varphi$
	$\Diamond^{s}_{(a,b)} \varphi$
	$\Diamond^{s}_{[a,b)} \varphi$
	$\Diamond^{s}_{(a,b]} \varphi$
	$\Diamond^{s}_{[a,b]} \varphi$
	$\Diamond^{s}_{>b} \varphi$
	$\Diamond^{s}_{\geq b} \varphi$
	$\Box^{w} \varphi$
	$\Box^{w}_{(a,b)} \varphi$
	$\Box^{w}_{[a,b)} \varphi$
	$\Box^{w}_{(a,b]} \varphi$
	$\Box^{w}_{[a,b]} \varphi$
	$\Box^{w}_{>b} \varphi$
	$\Box^{w}_{\geq b} \varphi$
	$\varphi_1 \mathcal{U}^{s} \varphi_2$
	$\varphi_1 \mathcal{U}^{s}_{(a,b)} \varphi_2$
	$\varphi_1 \mathcal{U}^{s}_{[a,b)} \varphi_2$
	$\varphi_1 \mathcal{U}^{s}_{(a,b]} \varphi_2$
	$\varphi_1 \mathcal{U}^{s}_{[a,b]} \varphi_2$
	$\varphi_1 \mathcal{U}^{s}_{>b} \varphi_2$
	$\varphi_1 \mathcal{U}^{s}_{\geq b} \varphi_2$
		Future temporal operators

Table 1. STL/PSL operators

STL/PSL	STL	Description
`once varphi`	$\diamondminus \varphi$	Past temporal operators
`once(a:b) varphi`	$\diamondminus_{(a,b)} \varphi$	
`once[a:b) varphi`	$\diamondminus_{[a,b)} \varphi$	
`once(a:b] varphi`	$\diamondminus_{(a,b]} \varphi$	
`once[a:b] varphi`	$\diamondminus_{[a,b]} \varphi$	
`once[>b] varphi`	$\diamondminus_{>b} \varphi$	
`once[>=b] varphi`	$\diamondminus_{\geq b} \varphi$	
`historically varphi`	$\boxminus \varphi$	
`historically(a:b) varphi`	$\boxminus_{(a,b)} \varphi$	
`historically[a:b) varphi`	$\boxminus_{[a,b)} \varphi$	
`historically(a:b] varphi`	$\boxminus_{(a,b]} \varphi$	
`historically[a:b] varphi`	$\boxminus_{[a,b]} \varphi$	
`historically[>b] varphi`	$\boxminus_{>b} \varphi$	
`historically[>=b] varphi`	$\boxminus_{\geq b} \varphi$	
`varphi1 since varphi2`	$\varphi_1 \mathcal{S} \varphi_2$	
`varphi1 since(a:b) varphi2`	$\varphi_1 \mathcal{S}_{(a,b)} \varphi_2$	
`varphi1 since[a:b) varphi2`	$\varphi_1 \mathcal{S}_{[a,b)} \varphi_2$	
`varphi1 since(a:b] varphi2`	$\varphi_1 \mathcal{S}_{(a,b]} \varphi_2$	
`varphi1 since[a:b] varphi2`	$\varphi_1 \mathcal{S}_{[a,b]} \varphi_2$	
`varphi1 since[>b] varphi2`	$\varphi_1 \mathcal{S}_{>b} \varphi_2$	
`varphi1 since[>=b] varphi2`	$\varphi_1 \mathcal{S}_{\geq b} \varphi_2$	
`rise(varphi)`	$\uparrow \varphi$	Events
`fall(varphi)`	$\downarrow \varphi$	
`distance(phi1,phi2,k)`	$\lvert \phi_1 - \phi_2 \rvert \leq k$	Template properties
`distance(phi1,phi2,k,t,T)`	$(\lvert \phi_1 - \phi_2 \rvert > k) \rightarrow \Diamond^{s}_{[0,t]} \Box^{w}_{[0,T-t]} \lvert \phi_1 - \phi_2 \rvert \leq k$	
`distance(varphi1,varphi2,t,T)`	$(\neg(\varphi_1 \leftrightarrow \varphi_2)) \rightarrow \Diamond^{s}_{[0,t]} \Box^{w}_{[0,T-t]} (\varphi_1 \leftrightarrow \varphi_2)$	

Table 2. More STL/PSL operators