Kalpa Publications in Computing
Volume 3, 2017, Pages 106–122
RV-CuBES 2017. An International Workshop on
Competitions, Usability, Benchmarks, Evaluation,
and Standardisation for Runtime Verification Tools

ARTiMon Monitoring Tool
The Time Domains

Nicolas RAPIN

CEA LIST
Bote Courrier 174, Gif sur Yvette, F-91191 France
nicolas.rapin@cea.fr

Abstract

This work is related to our monitoring tool called ARTiMon for the online property monitoring of discrete and continuous systems. This paper presents the syntax and the semantics of the current input language of ARTiMon. Because this is not always well understood by users, and also because it grounds the ability of ARTiMon to check properties online, in this tool paper a particular focus is accorded to the domain definition of time functions used to interpret terms of the language.

1 Introduction

Property monitoring is a unified solution in order to detect failures at many stages of a system's life-cycle. It can be applied online or offline. An online monitor is supplied with some system states at some times called *time stamps*. The online mode supposes the satisfaction of a real time constraint: from the time it is supplied with a state the monitor should perform and terminate an interpretation of the checked properties before the next state is supplied. An offline monitor is not constrained by such imperative deadlines. In contrast, it requires a complete trace to evaluate properties. It result that the offline analysis has many drawbacks depending on the level considered. At the exploitation level it is not suitable to control a running system. It can only deliver a *post-mortem* diagnostic. At the validation level, for example based on the simulation of some system models, the simulation trace must be completed before the analysis of properties starts. If the computation of a simulation trace is time consuming, the offline approach may represent a huge waste of time, especially when the failures (pieces of traces that validate hazard properties or violating expected invariants) occurs at the beginning of the trace. In order to get around those drawbacks while proposing a rich specification language, our tool, called ARTiMon (for Accurate Real Time Monitor), has been designed to evaluate online and in real-time terms of a rich language derived from real-time logics. At the design level it can be to used to check that simulation traces of systems models are conform to some properties. At the deployment level it can be used as an embedded monitoring agent for online diagnostic [1] or for controlling purposes. Whatever it is used for, ARTiMon notifies failures with the smallest possible lag. Thus a simulation can be stopped early in order to correct a wrong model, or the running system can be controlled to restore its conformity. Historically ARTiMon has been

developed for the online validation of Matlab/Simulink models with respect to real time properties. Its development started during the Eureka European project called SYSPEO. ARTiMon, at least a version of it, remains available for the analysis of the simulations of Simulink models. ARTiMon is directly embedded in Simulink models as a S-Function block.

In this paper, we present the current input language of ARTiMon (Version 0.9). Concerning the formal semantics, ARTiMon interprets all terms of its input language as partial time functions i.e. as functions whose domains are bounded intervals of a time domain which can be discrete or continuous depending on the nature of the system. Notice that the interpretation (or semantics) of a term is always defined with respect to a sequence of timed states, called a *trace*, which formalizes a system execution. A trace being a static object the semantics is indeed defined in an offline style. This rises a tension with the online context: considering a trace analyzed by an online monitor we must assume that the monitor has discovered this trace step by step, or we should we say state by state. It wouldn't be realistic to suppose that this online monitor re-computes completely the semantics at each new state supplied, as if the current trace prefix formed so far was a completely new fresh trace, especially because all those prefixes are longer and longer. We can then assume that a realistic online monitor, at each arrival of a new state, make a re-use of the semantics computed for the previous states. For short, to compute the semantics for the next trace prefix the online monitor uses, or let us say extends, the computation achieved for the current trace prefix. It results from this discussion that a trace can be considered as a static object but also, from a dynamical point of view, as a temporary prefix of a further trace extension. Interpretations of terms inherit of this character. They are partial time functions destined to be extended as and when the underlying trace is extended.

Deriving from those considerations, the online interpretation can be defined as the concatenation of partial, adjacent and local offline interpretations triggered by traces extensions. Online interpretation must give the same result as the offline one. To ensure this, we have to take care that an interpretation already performed with respect to a given trace σ, assigning a value to a term at a given time, cannot be contradicted (on times of its domain) by the interpretation for any trace σ' extending σ. An interpretation ensuring this will be called *sound* for σ. This interpretation should be also *complete* for σ, that is maximal with respect to all sound interpretations. Completeness ensures that all partial interpretations are adjacent and hence the continuity of their union. Those concerns explain why a large part of this paper is devoted to the definition of the sound and complete domain of the time function representing the interpretation (the semantics) of a term with respect to a trace as it is computed by the ARTiMon tool. The paper is organized as follows. In Section 2 we recall basic definitions about systems and traces. The Section 3 introduces the formal syntax of ARTiMon language. Section 4 details the semantics of this language. It begins with ground and atemporal terms and is followed by a description of the semantics for temporal terms. The short Section 5 describes how and when ARTiMon notifies failures. Section 6 gives some specification examples inspired by industrial collaborations. The Section numbered 7 evokes some implementation aspects, contributions in industrial projects and availability of the tool. Section 8 refers to some similar tools and we conclude this paper in Section 9.

2 System Under Monitoring, Traces

A system under monitoring (SUM for short) is characterized by a finite set V of *state variables* and by a time domain $\mathbb{T}$ which can be $\mathbb{Z}$ if the SUM is discrete or $\mathbb{R}$ if the SUM is continuous.

Each state variable has a type (like bool, int, float) which defines its *range*. The symbol $\perp$ will be used for denoting the false value of the boolean range $\{\perp, \top\}$ but also for denoting a special value called *the undefined value* for any other range. That is to say we assume that $\perp$ is present in any range of any state variable. This overloading of the symbol $\perp$ simplifies many definitions, especially the definition of negative intervals (see below).

A timed state ν is a tuple (t, s) where $t \in \mathbb{T}$ is a time stamp and s a state (i.e. a map assigning to each $v \in V$ a value of its range denoted by $s(v)$). A *trace* of the SUM is any finite sequence σ of timed states of the form $((t_0, s_0), \ldots, (t_n, s_n))$ being such that $(t_0, \ldots, t_n)$ is strictly increasing. The domain of σ, noted $|\sigma|$, is $[t_0, t_n]$. A trace σ' of the form $((t_0, s_0), \ldots, (t_n, s_n), (t_{n+1}, s_{n+1}))$ is called a *trace extension* of σ. We say that σ' induces a *domain extension* with respect to $|\sigma|$, it is the interval $]t_n, t_{n+1}]$. For short we have $|\sigma'| = |\sigma| \cup]t_n, t_{n+1}]$. It results that a given non-empty trace can be seen as a whole or as an extension of a trace or even as the result of successive extensions from the empty trace $\emptyset$.

As a running example, let us consider a SUM characterized by a boolean state variable b and by a float state variable v. We will consider three timed states $\nu_0 = (0, \{b \mapsto \perp, v \mapsto 1.25\})$, $\nu_1 = (2, \{b \mapsto \top, v \mapsto 1.25\})$ and $\nu_2 = (3, \{b \mapsto \perp, v \mapsto 1.65\})$ and the traces $\sigma_2 = (\nu_0, \nu_1, \nu_2)$ extending $\sigma_1 = (\nu_0, \nu_1)$ extending $\sigma_0 = (\nu_0)$ extending $\emptyset$. Notice that time stamps are not necessarily regularly spaced.

Considering a trace as a whole (perspective adopted by the offline monitoring techniques), we could say that property monitoring consists in evaluating some properties with regard to the trace. Now considering a trace as the result of a sequence of trace extensions, we can give a slightly different definition suitable for the online approach: property monitoring consists in concatenating partial interpretations triggered by each trace extension. Of course, whatever is the perspective chosen and its underlying interpretation technique, both must lead to the same result. As it will be exposed, ARTiMon is strongly based on this second point of view. That is why, as mentioned in the introduction, we need a rigorous definition of those partial interpretations to preserve an equality with the offline interpretation. As it will be exposed this relies on the notion of sound and complete time domains for the interpretation of a term.

3 Specification Language

The *ground terms* of our language are symbols of constants, represented by letter c, and state variable symbols, represented by letter v. If one combines those ground terms using functions, predicates, or build vectors from those terms, indeed he only generates additional state variables. That's why we call atemporal such terms. Formally we call *atemporal* the terms deriving from this BNF grammar:

$$\phi \ ::= \ c \mid v \mid g\phi \mid (\phi, \ldots, \phi)$$

The symbol g stands for a symbol of function. It may be $\neg, \wedge, \vee, \rightarrow$ to mention boolean ones or $*, +, -, \div, \ldots$ to mention float ones. A term of the form $g\phi$ denotes the application of function g to the term ϕ considered as an argument. This latter can be a *vector* of the form $(\phi_1, \ldots, \phi_n)$. Thus one can form for example the term $\wedge(\phi_1, \phi_2)$ which is the prenex form of the usual boolean conjunction usually noted $\phi_1 \wedge \phi_2$. We opt for the prenex form as it extends more naturally to functions whose arity is greater than 2. In an ARTiMon input file, a vector is declared as follows: $\langle vector\_alias \rangle[n]\ \phi_1\ \ldots\ \phi_n$. So n stands here for the dimension of the

vector. The Figure 1 of Section 7 gives an example with $n = 2$.

The language of ARTiMon extends those atemporal terms by introducing temporal operators. ARTiMon language is a subset of terms given by this BNF grammar (which extends the previous one):

$$\phi \ ::= c \mid v \mid g\phi \mid (\phi, \ldots, \phi) \mid A\{f\}_j \ \phi \mid \uparrow \ \phi \mid \downarrow \ \phi \mid \phi \ U_i \ \phi \mid$$

$$\phi \ S_k \ \phi \mid \phi \ D@ \ \phi \mid iL \ \phi \mid cL \ \phi \mid \phi \ Cut \ \phi \mid \phi \ O \ \phi \mid L \ \phi \mid P \ \phi$$

We said that ARTiMon language is a subset of terms defined by this grammar because terms should be consistent regarding their type. For example the term $\wedge(\top, 18.2)$ is not well-formed. This typing issue is not treated here but what is a well formed term is rather intuitive and left out here. Some of our temporal operators come along with an interval (in subscript) called its *scope* whose purpose is to restrict the quantification (or the aggregation) of the operator. Depending on the operator, the scope must satisfy some specific constraints. For example, the scope i of the binary until operator, noted U_i, is imperatively a positive bounded interval i.e. an interval whose bounds are positive bounded numbers. The scope k of the since operator, noted S_k, must have a positive or null low bound and the upper bound can be either a standard positive number or $+\infty$.

A term of the form $A\{f\}_j$ is called an *aggregation* term. It has been presented in [9] when j is bounded. Here we also accept that $j.lb = -\infty$. This means that one is allowed to aggregates values of a term from the beginning of its definition. In contrast with the languages of [8, 7] where non boolean signals are also introduced but indeed reduced to atemporal boolean terms using predicates, our aggregation operator allows the creation of non boolean terms that are temporal. Notice that the well-known bounded *eventually* operator usually noted $\Diamond_j$ can be emulated using aggregation: $\Diamond_j\phi$ stands for $A\{disj\}_j\phi$. The dual *globally* operator is noted $\Box_i$ and $\Box_i\phi$ stands for $\neg\Diamond_i\neg\phi$. $\uparrow$ and $\downarrow$ stands respectively for the rising and falling edge operators. ARTiMon extends their application to non-boolean terms as it will be explained below. The most non conventional operators are the *Duration At* operator noted $D@$, the *incremental length* operator noted iL, the *cumulative length* operator noted cL, the *last value* operator noted L, the *previous value* operator noted P and the *over* operator noted O. Their intuitive and formal semantics are detailed below in Subsection 4.2.

4 Semantics

As in [9] the semantics of any term ϕ with respect to a trace σ is a time function noted $(\phi)_\sigma$ whose domain, being a bounded interval of $\mathbb{T}$, is noted $|(\phi)_\sigma|$. Whatever are $\mathbb{T}$ and ϕ, the function $(\phi)_\sigma$ is always a finite union (concatenation) of constant functions whose domains are adjacent intervals of $\mathbb{T}$ (two intervals are *adjacent* if their union is an interval and their intersection is empty) covering $|(\phi)_\sigma|$.

4.1 Semantics of Ground and Atemporal Terms

Let us illustrate for our running example. Whatever is $\mathbb{T}$, the domains $|(v)_{\sigma_2}|$ and $|(b)_{\sigma_2}|$ are both equal to $|\sigma_2| = [0, 3]$. The function $(v)_{\sigma_2}$ is the union of those constant functions: $[0, 3[\to \{1.25\}$, $[3, 3] \to \{1.65\}$ and $(b)_{\sigma_2}$ is the union of: $[0, 2[\to \{\bot\}$, $[2, 3[\to \{\top\}$, $[3, 3] \to \{\bot\}$.

In the discrete semantics (i.e. $\mathbb{T}$ is $\mathbb{Z}$), the interval $[a, b[$ can also be written $[a, b-1]$. So, assuming this semantics, we could also define $(v)_{\sigma_2}$ as the union of: $[0, 2] \to \{1.25\}$, $[3, 3] \to \{1.65\}$. In the remainder of this paper we will focus more on the continuous semantics. So, unless said otherwise one can assume that $\mathbb{T} = \mathbb{R}$.

It is quite straightforward that a finite concatenation of constant functions can be represented (and implemented) by a list of valued time intervals. Let us formalize this now. A *time interval* is a tuple of the form (l, lb, ub, u) where $lb, ub \in \mathbb{T} \cup \{-\infty, +\infty\}$ are the bounds of the interval, and $l, u \in \{\top, \bot\}$ are boolean values defining the membership of bounds to the interval: lb (resp. ub) belongs to the interval iff $l = \top$ (resp. $u = \top$). For example $(\top, 1.51, 2.38, \bot)$ states for the interval usually noted $[1.51, 2.38[$. If $ub < lb$ the interval is considered as empty, *idem* if the constraint $(ub = lb) \wedge \neg(l \wedge u)$ is satisfied. We will use both notations, the usual bracketed one and the tuple one. Non relevant parameters will be replaced by symbol $-$. Thus $(\top, 0, -, -)$ denotes the interval containing 0 and potentially some greater time values. A valued interval is simply a 5-tuple (l, lb, ub, u, c) such that (l, lb, ub, u) is a time interval and c a constant of a given range. We may also use the notation $c^{(l,lb,ub,u)}$ to denote (l, lb, ub, u, c). Thus $1.25^{[0,3[}$ denotes $(\top, 0, 3, \bot, 1.25)$ itself representing the constant time function $[0, 3[\to \{1.25\}$. With respect to our last example, the function $(v)_{\sigma_2}$ would be represented by the list: $(1.25^{[0,3[}, 1.65^{[3,3]})$.

We call *negative* an interval whose value is $\bot$. For a boolean term it denotes a time interval on which it is false and for a non boolean term an interval on which it has no defined value. In order to save memory, ARTiMon only stores positive intervals (those whose value is not $\bot$) as negative ones are defined by the complement in the domain. Thus $(b)_{\sigma_2}$ whose domain is $[0, 3]$ would be represented by the list $(\top^{[2,3[})$ instead of $(\bot^{[0,2[}, \top^{[2,3[}, \bot^{[3,3]})$.

Notice that defining intervals as tuples is convenient to define some intervals constructions and notations. We will use pointed notation to denote intervals attributes. For example $i.ub$ denotes the upper bound of the time interval i. The opposite of i, noted $-i$ is $(i.u, -i.ub, -i.lb, i.l)$. Given the time value t and a time interval i, $t \oplus i$ denotes $(i.l, t + i.lb, t + i.ub, i.u)$ i.e. i with t as new origin or let us say i shifted of t. This operation extends from scalars to intervals i.e. $k \oplus i$ denotes $(k.l \wedge i.l, k.lb + i.lb, k.ub + i.ub, k.u \wedge i.u)$. Notations $t \ominus i$ and $k \ominus i$ abbreviate respectively $t \oplus -i$ and $k \oplus -i$. Given two time intervals i and k, we note $i \looparrowright k$ the set $\{t \mid t \oplus i \subset k\}$. Notice that $i \looparrowright k$ is $(i.l \to k.l, k.lb - i.lb, k.ub - i.ub, i.u \to k.u)$ (where $\to$ stands for the boolean implication, i.e. $a \to b$ states for $\neg a \vee b$). We define the partial order $\prec$ over intervals as follows: $i \prec j$ iff $i \cap j = \emptyset$ and $i.ub \leq j.lb$.

Time functions extend naturally to intervals: if (ϕ) is a time function and i an interval, $(\phi)(i) = \{(\phi)(t) \mid t \in i\}$. For example $(v)_{\sigma_2}([1, 3]) = \{1.25, 1.65\}$. Those values can be also chronologically ordered. Suppose that the restriction of function (ϕ) to interval i is the concatenation of constant functions $c_1^{i_1}, \ldots, c_n^{i_n}$ (where $i_l \prec i_{l+1}$ for $l \in [1, n-1]$) then $(\phi)^{seq}(i)$ denotes the sequence $(c_1, \ldots, c_n)$. Thus $(v)^{seq}_{\sigma_2}([1, 3]) = (1.25, 1.65)$.

Introduced above with an example, let us formalize the semantics of a given state variable. The idea is the following: any timed state of the trace assigns a value to this state variable and this assignment remains the same continuously until it is updated by a subsequent timed state. Formally, given a state variable v and σ a trace of the form $((t_0, s_0), \ldots, (t_n, s_n))$, $|(v)_\sigma| = |\sigma| = [t_0, t_n]$. For $t \in |(v)_\sigma|$, $(v)_\sigma(t)$ is the value given by the last timed-state occurring before or at t. Formally, let $t \in |(v)_\sigma|$, if there exists two successive timed states (t_i, s_i) and

(t_{i+1}, s_{i+1}) of σ such that $t_i \leq t < t_{i+1}$ then we have $(v)_\sigma(t) = s_i(v)$ either necessarily we have $t = t_n$ and then $(v)_\sigma(t) = s_n(v)$.

Though straightforward, we would like to emphasize two properties of this semantic definition: it is *sound* and *complete*. Sound because if one considers a trace extension σ' of σ it is clear that the function $(v)_{\sigma'}$ restricted to interval $|(v)_\sigma|$ is equal to $(v)_\sigma$ i.e. over $|(v)_\sigma|$ the semantic function $(v)_{\sigma'}$ agrees with (or does not contradict) $(v)_\sigma$. One could also says that this definition is such that a trace extension cannot induce a revision of previous values associated to the state variable and only induces a prolongation of the previous semantic time function. The definition is complete because its associated domain, being the domain of the trace, is maximal for soundness. To insist, notice also that if one considers a time $t' \notin |(v)_\sigma|$ then assigning a value for $(v)_\sigma(t')$ would be the result of an arbitrary choice i.e. this value would not depend on σ.

From a computation point of view soundness and completeness have a nice consequence: we deduce that $(v)_{\sigma'} = (v)_\sigma \cup \delta$ where δ is a time function over $|\sigma'| \setminus |\sigma|$. In other words, for computing $(v)_{\sigma'}$ we only have to compute δ and to merge this function with $(v)_\sigma$ to obtain $(v)_{\sigma'}$. δ is rather trivial for a state variable. With notations of Section 2, if $s_{n+1}(v) = s_n(v)$ then δ is $]t_n, t_{n+1}] \rightarrow s_n(v)$ else δ is the concatenation of the two constant time functions $]t_n, t_{n+1}] \rightarrow s_n(v), [t_{n+1}, t_{n+1}] \rightarrow s_{n+1}(v)$. The merging of two time functions like $(v)_\sigma$ and δ is quite straightforward. For this let us use their representations based on valued intervals lists. Let use suppose that $(v)_\sigma$ is represented by $i_0, \ldots, i_q$ and δ by $j_0, \ldots, j_r$. If i_q and j_0 are adjacent ($i_q.ub = j_0.lb$ and $i_q.u = \neg j_0.l$) and carry the same values ($i_q.c = j_0.c$) then $(v)_\sigma \cup \delta$ is represented by $i_0, \ldots, (i_q.l, i_q.lb, j_0.ub, j_0.u, i.c)\ , j_1, \ldots, j_r$ else by $i_0, \ldots, i_q, j_0, \ldots, j_r$.

What we just presented for ground terms applies identically to *atemporal* terms i.e. terms deriving from the sub-grammar $\phi ::= c \mid v \mid g\phi \mid (\phi, \ldots, \phi)$. Those terms can be considered as new state variables formed inside the language.

As it will be explained in Section 4.2, a fundamental aspect of our approach is to preserve those two properties, soundness and completeness, for the semantic definitions of all terms of our language in order to achieve semantics computation only over successive domain extensions. This is what we discuss now.

4.2 Semantics of Temporal Terms

As mentioned above, we conceive the online interpretation of the semantics of terms as the merging of partial interpretations corresponding to and triggered by all trace extensions resulting themselves from the acquisition of new timed states. For this, given a term ϕ the definitions of $|(\phi)_\sigma|$ (domain of (ϕ) w.r.t to the trace σ) and of $(\phi)_\sigma(t)$ (value of $(\phi)_\sigma$ at a given time t of $|(\phi)_\sigma|$) must be designed such that they can be extended (toward the future) with no contradiction. Formally those definitions must be sound and complete. Soundness means that for any term ϕ, considering any pair of traces σ', σ, where σ' is an extension of σ, we have $(\phi)_{\sigma'}(t) = (\phi)_\sigma(t)$ for all $t \in |(\phi)_\sigma|$. Completeness consists in choosing the maximal domain (with respect to intervals inclusion) compatible with soundness. Let us discuss this for each kind of terms.

4.2.1 Atemporal Combinations of Temporal Terms.

We have $|((\phi_1, \ldots, \phi_n))_\sigma| = \bigcap_{i\in[1,n]} |(\phi_i)_\sigma|$ and $|(g\phi)_\sigma| = |(\phi)_\sigma|$. Concerning the value definition at time t we have: $((\phi_1, \ldots, \phi_n))_\sigma(t) = ((\phi_1)_\sigma(t), \ldots, (\phi_n)_\sigma(t))$ and $(g\phi)_\sigma(t) = g((\phi)_\sigma(t))$.

4.2.2 Cut and Over Terms.

In the term $\phi = \phi_1\ Cut\ \phi_2$ the sub-term term ϕ_1 is assumed to be boolean. The type of such a term is the type of ϕ_2. It denotes the filtering of ϕ_2 with respect to the validity of ϕ_1. It is undefined (or false) when ϕ_1 is false else equal to ϕ_2. More formally: $(\phi)_\sigma(t)$ is equal to $(\phi_2)_\sigma(t)$ if $(\phi_1)_\sigma(t) = \top$ else $(\phi)_\sigma(t) = \bot$. Concerning its domain: $|(\phi)_\sigma| = |(\phi_1)_\sigma| \cap |(\phi_2)_\sigma|$. This term is a generalization of the boolean conjunction.

Now let us consider an Over term $\phi = \phi_1\ O\ \phi_2$ where ϕ_1 and ϕ_2 are non boolean terms of the same type. We have $|(\phi)_\sigma| = |(\phi_1)_\sigma| \cap |(\phi_2)_\sigma|$; $(\phi)_\sigma(t)$ is equal to $(\phi_1)_\sigma(t)$ if $(\phi_1)_\sigma(t) \neq \bot$ else is equal to $(\phi_2)_\sigma(t)$. This term merges the semantics of ϕ_1 and ϕ_2 with a prevalence of ϕ_1.

Notice that those two operators are indeed instances of the rule $g\phi$. For example the term $\phi_1\ Cut\ \phi_2$ could be also presented under the form $Cut(\phi_1, \phi_2)$. Here they are included explicitly in the language as they are not standard mathematical functions.

4.2.3 Rising/Falling Edge Terms.

Assuming $\mathbb{T} = \mathbb{R}$ let us consider the rising edge operator $\uparrow \psi$ with ψ boolean. To shorten notations let $k = |(\psi)_\sigma|$. Intuitively such a term is true at time t if ψ becomes true at t. From this intuitive definition one can derive several non equivalent formal semantics. With some of them the operator $\uparrow$ is idempotent i.e. $(\uparrow (\uparrow \psi))_\sigma = (\uparrow \psi))_\sigma$ holds but with others not. For example this operational semantics is idempotent: for each interval $\top^i$ of the representation of $(\psi)_\sigma$ one adds an interval $\top^{[i.lb,i.lb]}$ to $(\uparrow \psi)_\sigma$. In this case we have $|(\uparrow \psi)_\sigma| = (k.l, k.lb, k.ub, \bot)$. The bound $k.ub$ is excluded because the value of $(\uparrow \psi)_\sigma$ at time $k.ub$ depends of the values of $(\psi)_\sigma$ after $k.ub$ that are not yet defined with respect to σ. Another semantics, not idempotent, is the following: $(\uparrow \psi)_\sigma = \top$ if there exists an $\epsilon \in \mathbb{T}$, $\epsilon > 0$, such that (ψ) value is $\bot$ on $[t-\epsilon, t[$ and $\top$ on $]t, t+\epsilon]$. In this case we have $|(\uparrow \psi)_\sigma| = (\bot, k.lb, k.ub, \bot)$ because if t is the low bound $k.lb$ then $[t-\epsilon, t[$ is not defined and if is the upper bound $k.ub$ then $]t, t+\epsilon]$ is not defined. ARTiMon language, which is not really stabilized with regard to this question, will certainly, in its final version, propose several distinguished versions of this operator with different semantics.

ARTiMon could also propose alternative rising/falling edge operators setting a default initial value for its sub-term. For example the term $\uparrow_\bot \psi$ which assumes that ψ is initially false. This setting may have an impact on the domain. For example, with the second formalization (non idempotent), we would have $|(\uparrow_\bot \psi)_\sigma| = (\top, k.lb, k.ub, \bot)$ since ψ is supposed to be false on $[k.lb - \epsilon, k.lb[$.

ARTiMon also supports terms of the form $\uparrow \psi$ where ψ is not boolean. In this case, if ψ changes from value c to c' at time t then $(\uparrow \psi)(t) = c'$ i.e. the list for $(\uparrow \psi)$ will contain $c'^{[t,t]}$. So the operator $\uparrow$ could be named the *new value* operator and it has a defined value only when the value of its sub-term changes. Notice that at the same time t, the dual falling term $\downarrow \psi$ takes the abandoned value: $(\downarrow \psi)(t) = c$. Elsewhere (i.e. at any time where the sub-term remains constant) the value of such terms is $\bot$ (which stands for the undefined value). The alternative

operator noted $\uparrow_c \psi$ specifies a default initial value for ψ and thus enables to have a rising edge occurrence at the low bound of $|(\psi)_\sigma|$. We don't give some more details for the falling edge term $\downarrow \psi$ which is dual (it is true when ψ falls from true to false value).

Now let us consider briefly the case $\mathbb{T} = \mathbb{Z}$. For this case the semantics definition we propose is the following: $(\uparrow \psi)$ is true at $t \in \mathbb{T}$ if ψ is false at $t-1$ and true at t. In this case we have $|(\phi)_\sigma| = (k.l, k.lb+1, k.ub, k.u)$. The alternative operator $\uparrow_\perp$ satisfies: $(\uparrow_\perp \psi)$ is true at $|(\psi)_\sigma|.lb$ if $(\psi)_\sigma$ is also true at this time, and $(\uparrow_\perp \psi)$ is true at another time t if $(\phi)_\sigma$ is false at $t-1$ and true at t. Thus we have $|(\uparrow_\perp \psi)_\sigma| = |(\psi)_\sigma|$.

4.2.4 Aggregation with a Bounded Scope.

This term has been studied in [9]. The term ϕ we consider here is of the form $A_i\{f\}\psi$. Intuitively at time t the value of the term $\phi = A_i\{f\}\psi$ is the aggregation w.r.t to the aggregation function f of the chronological sequence of values taken by ψ over $t \oplus i$. This latter sequence is formalized by $(\psi)^{seq}(t \oplus i)$. Considering $\overline{f}$ the extension of f to a sequence of values as introduced in [9] we have: $(\mathcal{A}\{f\}_i\ \psi)_\sigma(t) = \overline{f}((\psi)^{seq}_\sigma(t \oplus i))$. Concerning $|(\phi)_\sigma|$, soundness imposes that $t \oplus i$ does not exceed (neither in the past or in the future) $|(\psi)_\sigma|$. Otherwise it would be like an attempt to aggregate undefined values. From completeness principle we deduce that any time t being such that $(t \oplus i) \subseteq |(\psi)_\sigma|$ should belong to $|(\phi)_\sigma|$. It results that $|(\phi)_\sigma| = \{t \in \mathbb{T} \mid (t \oplus i) \subseteq |(\psi)_\sigma|\}$. With our notations this interval is exactly denoted by $i \looparrowright |(\psi)_\sigma|$. As shown in [9] in order to compute $(A_i\{f\}\psi)_\sigma$ from $(\psi)_\sigma$ we iterate chronologically on intervals of $(\psi)_\sigma$ and for each we perform an operation called its *backward propagation*. More precisely if c^k is a valued interval belonging to the representation of $(\psi)_\sigma$ we aggregate $(k \ominus i) \cap |(\phi)_\sigma|$ with the value c into the list representing $(\phi)_\sigma$. With respect to our running example, considering the term $\Diamond_{]0,1]}b$ (which stands for $A\{disj\}_{]0,1]}b$), we have $|(\Diamond_{]0,1]}b)_{\sigma_2}| =]0,1] \looparrowright [0,3] = [0,2]$. Since function $(b)_{\sigma_2}$ is represented by $(\top^{[2,3[})$ we only aggregate, using the aggregation function $disj$, the interval $(\top^{[2,3[} \ominus]0,1]) \cap |(\Diamond_{]0,1]}b)_{\sigma_2}| = \top^{[1,3[} \cap [0,2] = \top^{[1,2]}$ to the (empty) list representing $(\Diamond_{]0,1]}b)_{\sigma_2}$. It results that $(\Diamond_{]0,1]}b)_{\sigma_2}$ is represented by the list containing only one interval $(\top^{[1,2]})$ with domain $[0,2]$.

Notice that time shifting can be emulated using aggregation. This can be done with a term ϕ of the form $A_{[ub,ub]}\{:=\}\ \psi$, where $:=$ stands for the *overwrite* aggregation $(:= (e,a)$ returns e whatever is the aggregate $a)$. This shifts (ψ) of $-ub$ in time since $(\phi)(t) = \dddot{=}((\psi)^{seq}(t \oplus [ub,ub]))$ $= := ((\psi)(t+ub)) = (\psi)(t+ub)$. Typically it can be used to compare a term with its own value considered with a constant shift in time.

4.2.5 Aggregation with an Unbounded Scope.

Here we deal with a term ϕ of the form $A_{(\perp,-\infty,ub,u)}\{f\}\ \psi$. As being not yet published let us introduce the intuitive semantics of this operator. Leaving aside the contribution of the parameter u, at time t the value of the term $\phi = A_{(\perp,-\infty,ub,u)}\{f\}\ \psi$ is the aggregation w.r.t to f of the chronological sequence of values taken by $(\psi)_\sigma$ before $t+ub$ (on its own domain). The parameter u specifies if the value of $(\psi)_\sigma$ at time $t+ub$ is considered or not for the aggregation. More formally it is the aggregation w.r.t to f of the chronological sequence of values taken by $(\psi)_\sigma$ over $(t \oplus (\perp,-\infty,ub,u)) \cap |(\psi)_\sigma|$. So $(\phi)_\sigma$, regarding our soundness constraint, is defined at time t if we have $(t \oplus (\perp,-\infty,ub,u)) \cap |(\psi)_\sigma| \neq \emptyset$ and if$(t \oplus (\perp,-\infty,ub,u))$ does not exceed $|(\psi)_\sigma|$ in the future i.e. $(t \oplus (\perp,-\infty,ub,u)) \subset (\perp,-\infty,|(\psi)_\sigma|.ub,|(\psi)_\sigma|.u)$. Thus we

have $|(\phi)_\sigma| = (|(\psi)_\sigma|.l \wedge u, |(\psi)_\sigma|.lb - ub, |(\psi)_\sigma|.ub - ub, u \to |(\psi)_\sigma|.u)$.

The term $A_{(\bot,-\infty,ub,u)}\{disj\}\ \psi$, with ψ boolean, which could be also noted $\Diamond_{(\bot,-\infty,ub,u)}\phi$ emulates a *before* term. It holds at a time t if ψ holds somewhere before $t + ub$ (or at $t + ub$ if u is $\top$).

4.2.6 Cumulative and Incremental Length Terms.

Due to our soundness principle, our approach is not compatible with unbounded future operators. Accepting them would lead us to speculate on undefined future values and this is not sound. Imagine a term of the form $\phi = A\{f\}_{(i.l,i.lb,+\infty,\bot)}\psi$. Its value at time t would depend on values of ψ over $t \oplus (i.l, i.lb, +\infty, \bot)$ which necessarily exceeds $|(\psi)|$, this latter being bounded. Another way to highlight this impossibility is to compute the domain respecting soundness. Since by induction $|(\psi)|$ is bounded (or $\emptyset$) then $|(\phi)|$ would remain $\emptyset$ since $(i.l, i.lb, +\infty, \bot) \nrightarrow |(\psi)| = \emptyset$. For a similar reason we cannot have an operator which would denote how long its sub-term will remain constant (or true) because implicitly such an operator would be an unbounded future operator. Instead we propose two alternative operators, the cumulative length and the incremental length. The intuitive idea of the cumulative length, whose associated term is of the form $cL\ \psi$ is to provide, when (ψ) remains constant with value c, the pair (c, l) where l is the delay elapsed from the time when (ψ) started to have value c. Notice we have $|(cL\ \psi)| = (|(\psi)|.l, |(\psi)|.lb, |(\psi)|.ub, \top)$ because the length of an interval is defined whatever its upper bound is included or not. More technically the pair (c, l) is defined each time a constant function composing (ψ) is extended. More formally considering $(\psi)_\sigma$ and $(\psi)_{\sigma'}$ where σ' is a trace extension of σ, if there exists c^i in the representation of $(\psi)_\sigma$ and $c^{i'}$ in the representation of $(\psi)_{\sigma'}$ such that $i \subset i'$ then we have $(cL\ \psi)(i'.ub) = (c, i'.ub - i.lb)$. At any other time of $t \in |(cL\ \psi)|$ the function has no defined value i.e. we have $(cL\ \psi)(t) = \bot$. In the same context the term $(iL\ \psi)(i'.ub)$ gives the size of the interval extension, hence we would have $(iL\ \psi)(i'.ub) = (c, i'.ub - i.ub)$.

Let us illustrate with our running example. First let us investigate how the function (v) evolves as and when timed states become available for the monitor. This amounts to study $(v)_{\sigma_0}$, $(v)_{\sigma_1}$, $(v)_{\sigma_2}$. We have $(v)_{\sigma_0} = (1.25^{[0,0]})$, $(v)_{\sigma_1} = (1.25^{[0,2]})$ and $(v)_{\sigma_2} = (1.25^{[0,3[}, 1.65^{[3,3]})$. One can see that the domain of the constant function 1.25 as been extended two times, from $[0, 0]$ to $[0, 2]$ and from $[0, 2]$ to $[0, 3[$. $(cL\ v)$ value is $\bot$ over $[0, 3]$ except at times 2 and 3 which are the upper bounds of the two extensions and we have: $(cL\ v)_{\sigma_2}(2) = (1.25, 2)$, $(cL\ v)_{\sigma_2}(3) = (1.25, 3)$. Similarly $(iL\ v)_{\sigma_2}$ value is $\bot$ over $[0, 3]$ except at times 2 and 3 and we have: $(iL\ v)_{\sigma_2}(2) = (1.25, 2)$, $(iL\ v)_{\sigma_2}(3) = (1.25, 1)$.

Notice that $(iL\ \psi)$ is a pair. One can apply the multiplication forming the term $*(iL\ \psi)$. This term has the dimension of an integral $(time * value)$. Applied to our running example we obtain $*(iL\ v)_{\sigma_2}(2) = *(1.25, 2) = 2.5$ and $*(iL\ v)_{\sigma_2}(3) = *(1.25, 1) = 1.25$. Now we can sum the past values of this term forming the term: $A\{sum\}_{]-\infty,0]}(*(iL\ v))$. For example at time 3 we have $(A\{sum\}_{]-\infty,0]}(*(iL\ v)))_{\sigma_2}(3) = 2.5 + 1.25 = 3.75$ which the integral of the ground term v over $[0, 3]$ with respect to σ_2.

4.2.7 Duration At Term.

Here we consider a term $\phi = \phi_1\ D@\ \phi_2$ where ϕ_2 is assumed to be a boolean event. This latter assumption means that $(\phi_2)_\sigma$ has the form of a Dirac function: it is false everywhere except at

some punctual times. In other words any positive interval of its interval representation is of the form $[t,t]$. For example ϕ_2 can be a rising edge term. We have $|(\phi)_\sigma| = |(\phi_1)_\sigma| \cap |(\phi_2)_\sigma|$. This term returns an amount of elapsed time. This amount is only defined when the event ϕ_2 occurs and if ϕ_1 is true or has a defined value. Then suppose ϕ_2 occurs at time t i.e. $(\phi_2)_\sigma(t) = \top$ and suppose that $(\phi_1)_\sigma(t) = c$ where c is not $\bot$. It follows that $\{t' \in \mathbb{T} \mid t' \leq t \text{ and } (\phi_1)_\sigma(t) = c\}$ is a non-empty interval of the form $(-, t_{min}, t, \top)$. Then $(\phi)_\sigma(t) = t - t_{min}$. Intuitively it is the amount of time for which ϕ_1 has remained constant (or true) before ϕ_2 occurs. That's why this term should be understood as the duration of steadiness of ϕ_1 when event ϕ_2 rises.

4.2.8 Until Term.

Here we consider a term $\phi = \phi_1 \; U_i \; \phi_2$ such that $i.lb$ and $i.ub$ are in $\mathbb{R}^+$ and $0 \notin i$. Classically, the definition of $(\phi)_\sigma(t)$ is the following: $(\phi_1 \; U_i \; \phi_2)_\sigma(t) = \top$ iff there exists $t' \in (t \oplus i)$ being such that $(\phi_2)_\sigma(t') = \top$ and $(\phi_1)_\sigma(t'') = \top$ for any $t'' \in [t, t'[$. A weak variant, not discussed here, could only require that $t'' \in]t, t'[$. For the domain definition one can be inspired by the argumentation developed for aggregation: for $(\phi)_\sigma$ being sound at t first we must have $(t \oplus i) \subset |(\phi_2)_\sigma|$; secondly from requirement $(\phi_1)_\sigma(t'') = \top$ for any $t'' \in [t, t'[$ we deduce that $(t \oplus (\top, 0, i.ub, \bot)) \subset |(\phi_1)_\sigma|$ must hold. It results that $|(\phi)_\sigma| = (i \leftarrow\!\!\shortmid |(\phi_2)_\sigma|) \cap ((\top, 0, i.ub, \bot) \leftarrow\!\!\shortmid |(\phi_1)_\sigma|)$.

Let us study the computation of this term. For this let us suppose that $(\phi_1 \; U_i \; \phi_2)_\sigma(t) = \top$ for a given time t. This means there exists a time $t' \in t \oplus i$ and u_1 a positive interval of $(\phi_1)_\sigma$ such that $[t, t'[\subset u1$ and also a positive interval u_2 of $(\phi_2)_\sigma$ such that $t' \in u_2$. Introducing $u_1' = u_1 \cup \{u_1.ub\} = (u_1.l, u_1.lb, u_1.ub, \top)$, the condition $[t, t'[\subset u1$ is equivalent to $[t, t'] \subset u_1'$. So $t' \in u_1' \cap u_2$. Thus $u_1' \cap u_2$ is not empty. We can conclude that u_1 of $(\phi_1)_\sigma$ and u_2 of $(\phi_2)_\sigma$ can contribute to a positive interval of $(\phi_1 \; U_i \; \phi_2)_\sigma$ only if $(u_1 \cup \{u_1.ub\}) \cap u_2 \neq \emptyset$. Suppose this is the case, let us call K this intersection. Then $(\phi_1 \; U_i \; \phi_2)_\sigma$ has a truth value at time t due to u_1 and u_2 if $t \in u_1$ and $(t \oplus i) \cap K \neq \emptyset$. That is to say $t \in u_1 \cap (K \ominus i)$. So $u_1 \cap (K \ominus i)$ is the contribution for $(\phi_1 \; U_i \; \phi_2)_\sigma$ due to u_1 and u_2 if $K = u_1' \cap u_2$ is not empty.

4.2.9 Since Term.

The scope of a Since term can be bounded or not. A bounded Since term is any term ϕ of the form $\phi_1 \; S_i \; \phi_2$ where the scope i is such that $i.lb$ and $i.ub$ are in $\mathbb{R}^+$ and $0 \notin i$. Such terms are the past symmetrical of Until terms: $(\phi_1 \; S_i \; \phi_2)_\sigma(t) = \top$ iff there exists $t' \in (t \ominus i)$ being such that $(\phi_2)_\sigma(t') = \top$ and $(\phi_1)_\sigma(t'') = \top$ for any $t'' \in]'t, t]$. Notice that ARTiMon also proposes a weak variant WS_i requiring only $t'' \in]'t, t[$ which is convenient to express consistent properties that would turn to antilogies if expressed with the S_i operator. Typically the expression $granted \rightarrow (\neg \; granted \; WS_i \; valid\_ident)$, specifying that an access is *granted* (assumed to be an event, for example $\uparrow \; access$) at most once for a valid identification occurring within $-i$ before, would be an antilogy using the S_i operator. The domain for S_i is derived as for the Until operator. An unbounded Since is a term $\phi = \phi_1 \; S_{(l,lb,\infty,\bot)} \; \phi_2$ where $i.lb \in \mathbb{R}^+$ and such that $0 \notin (l, lb, \infty, \bot)$. Intuitively this term is true at time t if there exists a time t' before $t - lb$ where ϕ_2 is true and if ϕ_1 has remained true since t'. Formally we have: $(\phi_1 \; S_{(l,lb,\infty,\bot)} \; \phi_2)_\sigma(t) = \top$ iff there exists $t' \in (t \ominus (l, lb, \infty, \bot))$ such that $(\phi_2)_\sigma(t') = \top$ and $(\phi_1)_\sigma(t'') = \top$ for any $t'' \in]t', t]$. Again the weak version requires only $t'' \in]t', t[$. Now let us determine the domain. Soundness at time t requires soundness of t'. With regard to the definition we must have $t' \in |(\phi_2)_\sigma|$ and also $]t', t] \subseteq |(\phi_1)_\sigma|$. This latter constraint is equivalent to $[t', t] \in (\top, |(\phi_1)_\sigma|.lb, |(\phi_1)_\sigma|.ub, |(\phi_1)_\sigma|.u)$. We deduce that we must have $t' \in |(\phi_2)_\sigma| \cap (\top, |(\phi_1)_\sigma|.lb, |(\phi_1)_\sigma|.ub, |(\phi_1)_\sigma|.u)$. To shorten notations let us call K this latter intersection which represents the sound interval for t'. Now

our term is sound at time t if obviously $t \in |(\phi_1)_\sigma|$ and if $t \ominus (l, lb, \infty, \perp)$ is sound with regards to t' soundness. This amounts to have those two conditions satisfied: (1) $(t \ominus (l, lb, \infty, \perp)) \cap K \neq \emptyset$ (one walking backward from t can reach at least one sound time t' in the scope) and (2) $(t \ominus (l, lb, \infty, \perp))$ does not exceed K in the future (one walking backward from t can reach only sound times t' in the scope). Hence $|(\phi)| = (K.l \wedge l, K.lb + lb, K.ub + lb, l \rightarrow K.u) \cap |(\phi_1)_\sigma|$.

4.2.10 Last Value Term.

A last value ϕ term is of the form $L\ \psi$ where ψ is not boolean. Intuitively $(\phi)_\sigma(t)$ is the last defined value of $(\psi)_\sigma$ i.e. $(\phi)_\sigma(t) = c$ if there exists $t' \leq t$ such that $(\psi)_\sigma(t') = c$ and for all $t'' \in]t', t]$ we have $(\psi)_\sigma(t'') \in \{c, \perp\}$. Concerning its domain we have $|(\phi)_\sigma| = |(\psi)_\sigma|$. To compute $(\phi)_\sigma$ is suffices to copy $(\psi)_\sigma$, to restore negative intervals by complementation, and finally to replace any pair of adjacent valued intervals of the form $c^i, \perp^{i_{k+1}}$ by the single interval $c^{i \cup i_{k+1}}$.

4.2.11 Previous Value Term.

Despite it may appear counter-intuitive and less natural than in discrete time, we propose a previous value term which is also defined for the continuous semantics even with variable step traces. This term is of the form $\phi = P\psi$. Its semantics is more easy to describe using interval based representations. It consists simply in shifting the constant values in the list and to remove the first interval. Thus if $(\psi)_\sigma = (c_0^{i_0}, \ldots, c_n^{i_n})$ then $(P\psi)_\sigma = (c_0^{i_1}, \ldots, c_{n-1}^{i_n})$. Suppose for example that $t \in i_1$ then $(P\psi)_\sigma(t) = c_0$ while at the same time $(\psi)_\sigma(t) = c_1$. So $(P\psi)_\sigma(t)$ refers to the previous value of ψ since, at time t, c_0 is indeed the previous value taken by ψ relatively to its actual value c_1. Domain is preserved: $|(\phi)_\sigma| = |(\psi)_\sigma|$.

Such term is convenient to compare the value of a term with its own previous value. For example $<(P\psi, \psi)$ is true if ψ is increasing.

5 Online Notification

Each time the trace is extended, ARTiMon propagates this extension to the semantic functions of all terms which are computed over their own domain extension. In particular this is done for hazards (set of terms that should remain false). If an hazard term becomes true on its domain extension, ARTiMon rises a notification for this fact immediately.

6 Specification Examples

Notice that in examples given below, contrarily to the convention adopted in [9] we use the symbol $\perp$ instead of $\emptyset$ in the definitions of the aggregation functions. Though we should use prenex notations to be strictly compliant with our BNF grammar of Section 3, for readability we may use (and even mix both) standard and prenex notations. For example the product of a and b may be noted $a * b$ or in its equivalent prenex from $*(a, b)$. Notice that a term of the form $\div(A_{[-60,0]}\{min\_max\}\ freq)$ is well formed because the aggregation function min_max aggregates a value x into a pair (m, M) as follows: $min\_max(x, \perp) = (x, x)$, $min\_max(x, (m, M)) = (min(x, m), max(x, M)))$. So at a time t the aggregation term $A_{[-60,0]}\{min\_max\}\ freq$ is either undefined (equal to $\perp$) or a pair (m, M) where m (resp. M) is the min (resp. max) value of the frequency state variable $freq$ over the interval $t \oplus [-60, 0]$.

Hence the value of our term at t is either undefined (assuming $\div(\bot) = \bot$) or $\div(m, M)$ which is simply, using standard notations, the ratio $m \div M$. In the examples below final requirement may be decomposed using intermediate named terms.

Jogger Heart Frequency. The heart frequency $freq$ should be under 55 bpm if the jogger has not made any effort (pedometer *pedo* remains false for the last 300 time units). After a rest of 60 time units the heart frequency should have decrease of 30%.

$req_1 = (\Box_{[-300,0]}(\neg\ pedo)) \rightarrow (freq \leq 55)$.

$req_2 = (\uparrow (\Box_{[-60,0]}(\neg\ pedo))) \rightarrow ((\div(A_{[-60,0]}\{min\_max\}\ freq)) < 0.7)$

Mold Temperature *(Inspired from BEinCPPS European Project in collaboration with Pernoud S.A.)* The temperature *temp* of the mold should not exceed more than 5% of its average value considered over the last 10 time units.

$area = A_{[-10,0]}\{sum\}(*\ iL(temp))$

$average = area \div 10$

$requirement = temp \leq (1.05\ *\ average)$

PID Controller Performance. When the instruction r is steady (variation lower then %1) for 40 time units the gap between the measure y and r must be lower than 0.05.

$r\_steady = (\div(A_{[-40,0]}\{max\_min\}\ r)) < 1.01$

$requirement = r\_steady \rightarrow (abs(r - y) < 0.05)$

Consumption per Cycle *(Inspired from a collaboration with Sherpa Eng.)* The integral of the consumption *cons* (float) per *cycle* (identifier being an integer) should remain under 10. In this example we use the aggregation function *sum_reset* satisfying: $sum\_reset(a, \bot) = a$, $sum\_reset(a, b) = a + b$ if $a \neq MAX\_FLOAT$, $sum\_reset(MAX\_FLOAT, b) = 0$. With this function, aggregating the maximum float value to a sum reset it to zero. With this trick we can reset the computed integral each time a cycle ends (and a new one starts). The atemporal function max_float_conv used converts a boolean signal to a sum reset signal. It satisfies: $max\_float\_conv(\top) = MAX\_FLOAT$, $max\_float\_conv(\bot) = \bot$.

$cycle\_ends =\ \uparrow\ cycle$

$reseter = max\_float\_conv(\ cycle\_ends)$

$cons\_within\_cycle = (\neg\ cycle\_ends)\ Cut\ cons$

$reseter\_over\_elemtary\_surface = reseter\ O\ (*(LI\ cons\_within\_cycle))$

$integral\_per\_cycle = A\{sum\_reset\}_{]-\infty,0[}\ reseter\_over\_elemtary\_surface$

$requirement = integral\_per\_cycle \leq 10$

Online Adder 1. Its boolean output *adding* shall be true if and only if its inputs values $x1$ and $x2$ are constant for at least 3 time units. If *adding* holds then its output *val* shall be equal to $x1 + x2$. For the steadiness of inputs we use the cumulative length operator.

$x1\_constant = (cL\ x1) \geq 3$

$x2\_constant = (cL\ x2) \geq 3$

$add\_req = (x1\_constant \wedge x2\_constant) \leftrightarrow adding$

$sum\_req = adding \rightarrow ((x1 + x2) == val)$

$requirement = add\_req \wedge sum\_req$

Notice that since functions are represented by lists of positive intervals, one can overload the application of some boolean operators to non boolean terms. The value of a positive interval is then considered as an interval carrying the value $\top$. For example, an (almost) equivalent term for $(cL\ x) \geq 3$ is $\Box_{[-3,0]}x$. Indeed those two terms are not strictly equivalent since cL is not affected by bounds closure contrarily to $\Box_i$ (for example, for a given time t, the interval $[t-3, t[$ of (x) induces the validity of $(cL\ x) \geq 3$ at t while $\Box_{[-3,0]}x$ is not valid at t).

Here is an equivalent variant of the Adder specification, with less terms, hence more efficient to compute, based on the reuse of the pair $(x1, x2)$:

$x1\_x2\_vec = (x1, x2)$

$add\_req = ((cL\ x1\_x2\_vec) \geq 3) \leftrightarrow adding$

$sum\_req = adding \rightarrow (+\ x1\_x2\_vec == val)$

$requirement = add\_req \wedge sum\_req$

Online Adder 2. On event *request* (coming from a client) if at this moment its input values $x1$ and $x2$ are constant for at least 3 time units, its output value *val* should be equal to $x1 + x2$.

$requirement = (((x1, x2)\ D@\ request) \geq 3) \rightarrow (val == +(x1, x2))$

With respect to all those examples, obtaining an hazard to monitor with ARTiMon consists basically in adding the negation to the expression called *requirement* which is, for each example, an invariant i.e. an expression that should remains constantly true if the SUM executes correctly.

7 Implementation, Usage, Contributions in Industrial Projects, Distribution

ARTiMon (acronym for *A*ccurate *R*eal *Ti*me *Mon*itor) is implemented in C language and consists in three libraries. The first is dedicated to the handling of time and ensures that all arithmetic operations achieved on time values are correct. This justifies the *A* for Accurate, starting the name ARTiMon. Notice that such a library is necessary to ensure the conformity of the implementation of ARTiMon to its theoretical foundations partially presented here. For example we could not have use the double type of the standard C language as the operation $+$, necessary for example to compute $t \oplus i$, applied on doubles, is not commutative neither associative and even more is absorbing when one operand is drastically smaller compared to the other. A second library is used to handle and extend the *user part* of the language which consists in the set of atemporal functions (like g in the BNF rule $g\ \phi$) and aggregation functions

(like f in the rule $A\{f\}_i\phi$). The third and core library is in charge of computing online the semantics of terms and of notifying the satisfaction of terms representing hazards.

An input specification file for ARTiMon is simply a text file respecting a syntax. The Figure 1 below is a screeshot of such an input file (using the overload of globally instead of the cL operator). It corresponds to the Adder example given above. It is divided in four sections, separated by the symbol $*$. In the first one, one declares the boolean states variables (one per line) and in the second the non boolean state variables (one per line). In the third section one declares intermediate expressions with an alias. This alias is used as a reference to the expression in further declarations. The last and fourth section is dedicated to hazards which are terms that should remain false. ARTiMon provides a simple API to initialize some parameters and to charge the specification file, to refresh the state variables values and the current time. The state variables values are stored in an array (called *state variable array* in the remainder) handled in memory by ARTiMon. Refreshing a state variable just overwrites its value in this state variable array at an index associated to this variable. Only a call to the time refresh function is interpreted by ARTiMon as the availability of a new timed state. The value of its time-stamp is the value passed to this refresh time function. The associated system state is given by the current values stored in the state variable array. Thus, if a state variable is updated several times before the current time is updated, only the last update of this variable is taken into account (others are considered as non relevant). This new timed state extend the trace and then, as explained in previous sections, one can extends the semantic time functions associated to all terms. That's why a call to the refresh time function triggers the computation of the extensions of the semantic time functions of all terms. If an hazard expression becomes true on its own extension ARTiMon notifies this fact (by switching a flag, by dumping intervals in a report file, etc). Subsequently this also triggers the garbage collection mechanism, not discussed here, which ensures that the memory of ARTiMon remains bounded. The idea is to remove (or more exactly to recycle in order to avoid re-allocation) inductively, starting from hazard expressions, the intervals allocated for sub-terms that cannot contribute any more to validate any hazard over any of its extensions. The whole ARTiMon monitoring process we just described is formalized in pseudo-code by the Algorithm 1 below. The file *artimon.conf* which is read first, contains some parameters like the start time (time-stamps are considered only if are greater than this start time), the path of the property file, the garbage collection activation option, the name of the report file (if used), the verbosity level for the verdict file. Notice that ARTiMon is passive. One has to to call the API to refresh state variables and current time when it is relevant. For example, when ARTiMon is embedded as an S-function in a Simulink model, the wrapper calls those update functions only when a so-called *main step* (cf Simulink Terminology) occurs. In other applications ARTiMon refresh functions are called regularly using a timer of the processor. As an FMU component they are called when the orchestrator passes the control to this component.

An Xtext/Eclipse based editor is currently under development to enable user friendly expressions like proposed by the STIMULUS tool of ARGOSIM [10]. The idea here is to provide one or more expansions for the syntax of each operator in such a way that expressions look like natural language expressions. For example an expansion for $\phi \rightarrow \psi$ could be $if\ \phi\ then\ \psi$. Predicate $>$ would be expanded into *exceeds*. The term $\Box_i\ \phi$ would be expanded into ϕ *remains true over* i. Combining all those expansions one could form an expression like: if (*temperature exceeds threshold*) *remains true over* $[-3, 0]$ *then alarm*. An integration of the ARTiMon approach to the UML/SysML Papyrus Modeling Tool [6] is also under devel-

```
adding
*
x1
x2
val
*
false = 'bool_cst' 0    // constant false
tab[2] x1 x2            // tab is the pair (x1,x2)
x1_x2_sum = '+' tab     // x1_x2_sum  is x1 + x2
comp[2] x1_x2_sum val   // comp is the pair (x1_x2_sum, val)
x1_x2_steady = (G[-3,0] x1) & (G[-3,0] x2) // x1, x2 steady for 3 t.u.
equal = ('==' comp)     // sum == val
add_req = x1_x2_steady -> adding
sum_req = adding -> equal
*
haz_add = ! add_req
haz_sum = ! sum_req
```

Figure 1: The Adder property monitoring file: Adder.prop.

opment. Some wrappers have been developed in order to adapt ARTiMon to many contexts. We developed a wrapper to embed ARTiMon in Simulink models as S-Function blocks in order to couple the monitoring of properties with the simulation process. ARTiMon computes the semantics prolongation only when a *main step* occurs (cf Simulink terminology). Thus we avoid the problem of handling rollbacks. The development of this tool has been started for the SYSPEO project [11] and has been continued through collaborations with Sherpa Engineering which is a massive user of the MATLAB/Simulink environment for its business. An ARTiMon wrapper for the FMU standard as also been developed and used to analyze simulations of nuclear plant components in interaction with the ALICE simulator from CORYS and with some FMU exports stemming from the Scade tool from Esterel Technologies. This latter work has been supported by the French BGLE project called CONNEXION [3]. For the case study with Pernoud S.A. mentioned in Section 6 ARTiMon is running on a Beable Bone Black on which a temperature captor is plugged in order to measure the mold temperature.

The licensing for the ARTiMon tool is not yet precisely defined. Until now it has been mostly used in the context of *R&D* collaborative projects. The author can transmit any license request to the CEA LIST institute which is has the ability to deliver licenses.

8 Similar Tools

To our knowledge most of property monitoring tools handle either a discrete or a continuous semantics but not both. Among continuous ones the most close tool to ARTiMon is certainly AMT [8] which is also based on the interval based representation of continuous functions having a bounded variability. In contrast with ARTiMon, AMT handles their computation offline. For discrete systems the tool LOLA [4] is able to monitor systems online with respect to a language enabling the expression of temporal logic properties and interesting statistical measures. The tool Temporal Rover [5] proposes a language combining linear temporal logic and real time temporal logic for the online monitoring of discrete systems. Let us also mention the recent commercial tool STIMULUS of ARGOSIM [10] which is able to compile some temporal assertion

Algorithm 1 ARTiMon Monitoring Process/Thread

```
1: Parsing of the artimon.conf file
2: Parsing of the properties file // we call H be the set of hazards declared
3: while (true) do
4:     while No New Time Stamp do
5:         State Variables Updates
6:     end while
7:     Computation of the Prolongation of the Semantic Time Functions of All Terms
8:     for φ ∈ H do
9:         if φ is true on its Prolongation then
10:            Notification of φ Surgery (socket message, flag positioning, file generation, ...)
11:        end if
12:    end for
13:    Garbage Collector
14: end while
```

into C observers and proposes a user friendly interface to express property based specifications. The tool LARVA [2] is suited for the monitoring of JAVA code execution.

9 Conclusion

In this tool paper we presented the current input language of ARTiMon and its semantics. This language aims the specification of rich and complex properties about traces of real-time running systems. We focused on the fact that fundamentally the semantics of any term is a time function. Aiming an online computation of this semantics we attached us to determine the domain of such a time function under the online perspective. This perspective consists in considering that a system under monitoring is not supposed to stop, and as a consequence that any finite trace of such a system is intended to be extended. From this idea we deduced that the interpretation of a term with respect to a given trace, defined as a time function, is also a partial function intended to be extended regarding its domain, as and when the trace is itself extended. In this paper we presented how this perspective induces a sound and complete bounded domain for the interpretation of each kind of terms. Details on the operational computation have not been presented exhaustively, especially for operators having an unbounded scope. Some variants for some operators have been discussed or just evoked, showing that ARTiMon language could evolve in the future. Especially some variants about until and since operators could be more developed. All those details of computation, variants of some operators, and complexity in time and space, should be delivered in further publications. We also left aside a detailed presentation of the garbage collection mechanism used by ARTiMon to ensure that the memory of a monitor remains bounded in time. This question, addressed in [9] for the aggregation term should be also developed in forthcoming publications for the whole terms.

References

[1] Michel Batteux, Philippe Dague, Nicolas Rapin, and Philippe Fiani. Diagnosability study of technological systems. In Kishan Mehrotra, Chilukuri Mohan, Jae Oh, Pramod Varshney, and Moonis Ali, editors, *Modern Approaches in Applied Intelligence*, volume 6703 of *Lecture Notes in*

Computer Science, pages 186–198. Springer Berlin / Heidelberg, 2011. 10.1007/978-3-642-21822-4_20.

[2] Christian Colombo, Gordon J. Pace, and Gerardo Schneider. Larva — safer monitoring of real-time java programs (tool paper). In *Proceedings of the 2009 Seventh IEEE International Conference on Software Engineering and Formal Methods*, SEFM '09, pages 33–37, Washington, DC, USA, 2009. IEEE Computer Society.

[3] French BGLE Project CONNEXION. https://www.cluster-connexion.fr/.

[4] Ben D'Angelo, Sriram Sankaranarayanan, César Sánchez, Will Robinson, Bernd Finkbeiner, Henny B. Sipma, Sandeep Mehrotra, and Zohar Manna. Lola: Runtime monitoring of synchronous systems. In *12th International Symposium on Temporal Representation and Reasoning (TIME'05)*, pages 166–174. IEEE Computer Society Press, June 2005.

[5] Doron Drusinsky. The temporal rover and the atg rover. In Klaus Havelund, John Penix, and Willem Visser, editors, *SPIN Model Checking and Software Verification: 7th International SPIN Workshop, Stanford, CA, USA, August 30 - September 1, 2000. Proceedings*, pages 323–330. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[6] PAPYRUS UML/SysML Modeler. https://eclipse.org/papyrus.

[7] Dejan Nickovic. *Checking Timed and Hybrid Properties: Theory and Applications*. PhD thesis, University Joseph Fourier, 2008.

[8] Dejan Nickovic and Oded Maler. Amt: A property-based monitoring tool for analog systems. In Jean-François Raskin and P. S. Thiagarajan, editors, *Formal Modeling and Analysis of Timed Systems: 5th International Conference, FORMATS 2007, Salzburg, Austria, October 3-5, 2007. Proceedings*, pages 304–319. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[9] Nicolas Rapin. Reactive property monitoring of hybrid systems with aggregation. In Yliès Falcone and César Sánchez, editors, *Runtime Verification: 16th International Conference, RV 2016, Madrid, Spain, September 23–30, 2016, Proceedings*, pages 447–453. Springer International Publishing, Cham, 2016.

[10] STIMULUS. http://argosim.com/product-overview.

[11] European Eureka Project SYSPEO. http://www.eurekanetwork.org/project/id/3796.