Institute of Science and Technology

Model Measuring for Hybrid Systems

Thomas A. Henzinger and Jan Otop

Technical Report No. IST-2014-171-v1+1
Deposited at 19 Feb 2014 10:16
http://repository.ist.ac.at/171/1/report.pdf

IST Austria (Institute of Science and Technology Austria)
Am Campus 1
A-3400 Klosterneuburg, Austria

Model Measuring for Hybrid Systems

Thomas A. Henzinger and Jan Otop*

Institute of Science and Technology Austria (IST Austria)

February 18, 2014

Abstract

As hybrid systems involve continuous behaviors, they should be evaluated by quantitative methods, rather than qualitative methods. In this paper we adapt a quantitative framework, called model measuring, to the hybrid systems domain. The model-measuring problem asks, given a model M and a specification, what is the maximal distance such that all models within that distance from M satisfy (or violate) the specification. A distance function on models is given as part of the input of the problem. Distances, especially related to continuous behaviors are more natural in the hybrid case than the discrete case. We are interested in distances represented by monotonic hybrid automata, a hybrid counterpart of (discrete) weighted automata, whose recognized timed languages are monotone (w.r.t. inclusion) in the values of parameters.

The contributions of this paper are twofold. First, we give sufficient conditions under which the model-measuring problem can be solved. Second, we discuss the modeling of distances and applications of the model-measuring problem.

1 Introduction

Hybrid systems combine discrete control and continuous dynamics. Hybrid automata, which are used to model hybrid systems, specify discrete control and continuous behaviors by, respectively, finite automata and continuous functions parametrized by time. These two aspects of hybrid automata influence each other. The current state of the discrete control determines a system of differential inequalities that specify the evolution of continuous functions. The transitions of the discrete control are labeled by linear inequalities, called guards. A transition is enabled only if its guards are satisfied by the current values of continuous functions.

Hybrid automata allow for the precise modeling of physical features. Consider a model of a water-level monitor depicted in Figure 1. The system monitors the level of water in a tank by switching the pump on and off to keep the water level between 0 and 13 units; it works as follows. If the pump is on, the water level rises constantly at pace 1 units per time unit. When the pump is off, the water level drops constantly at pace 2. Finally, there is a latency of (at most) 2 time units between the decision of the monitor to shut down (turn on) the pump and the effect. The functions t, l represent a clock and the water level.

However, while modeling of the system's physical part, the designer has to come up with numerical bounds occurring in guards, like the aforementioned latency of the pump. Those values are often estimated by the designer. In consequence, guards are often too restrictive. In the water-level monitor example, the latency of the pump can be safely increased from 2 to 2.5. That issue can be avoided by employing automatic parameter synthesis, which can be expressed as an instance of the model-measuring problem.

The *model-measuring problem* asks, given a model M and specification P, what is the maximal distance ρ such that all models M' within that distance from M satisfy (or violate) P. That distance ρ is called the *stability radius*.

*The second author is on the leave from University of Wrocław.

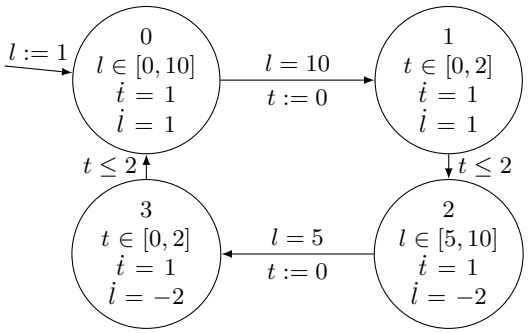
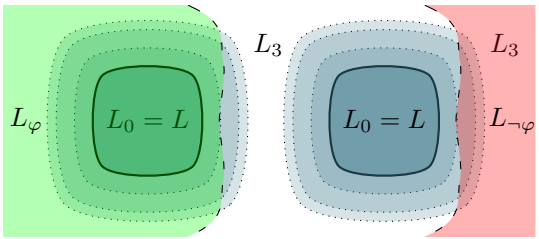


Figure 1: The water-level monitor from [1].



(a) Containment (b) Disjointness

Figure 2: A similarity measure and sets of traces satisfying (a) and violating (b) the specification. The model checking problem reduces to containment (a) and disjointness (b).

To determine the stability radius, it suffices to have a unary function that, for a given transition system M', specifies its distance from M. Such a function, called a *similarity measure*, is an input to the model-measuring problem. As inputs are required to be finitely represented, we are interested in *automatic* similarity measures that are represented by parametrized hybrid automata in the following way.

Consider a parametrized hybrid automaton $\mathcal{A}[p]$ with a single parameter p, which occurs in its guards. E.g. a parameter p in $x < p$. Assume that with p instantiated to 0, $\mathcal{A}$ recognizes precisely the set of traces of M and the timed languages recognized by $\mathcal{A}$ are monotonic (w.r.t. inclusion) in the values of the parameter p. Such a hybrid automaton defines a monotonic function from $[0, \infty]$ to the set of timed languages (ordered by inclusion), $x \mapsto \{L_x\}$, where L_x is the language of $\mathcal{A}$ with p instantiated to x (cf. Figure 2). Then, $\mathcal{A}$ defines a similarity measure in the following way, the distance of M' from M is the minimal value x such that L_x contains all traces of M'. This definition of a similarity measure allows for effective computation of the stability-radius.

The model satisfies the specification if its set of traces is contained in the language of all admissible traces. In consequence, the stability radius in the model-measuring problem is the supremum of the values x of the parameter p such that L_x is contained in the language of the specification (Figure 2.a). Indeed, the set of traces of each model whose distance from M is less then x is contained in L_x. On the other hand, maximality of x implies that x is the maximal distance.

However, if the specification L_P is given as the complement of the set of all valid traces, the model checking problem reduces to disjointness. Similarly, the model-measuring problem reduces to finding the supremum of the values of x of the parameter p such that L_x and L_P are disjoint (Figure 2.b). Finding that supremum, in turn, reduces to finding the infimum of values of the parameter p such that a given parametrized hybrid automaton with the parameter instantiated to this value recognizes a nonempty language. That problem for discrete weighted automata is referred to as the quantitative emptiness problem or the optimal-weight problem.

We discuss the optimal-weight problem for parametrized hybrid automata in Section 3. We give sufficient conditions, which guarantee that a class of parametrized hybrid automata admits an effective

procedure for approximating the infimum of the values of parameters, such that the hybrid automaton with parameters instantiated to these values recognizes a nonempty language.

Having an algorithmic foundation for the model-measuring problem, we can focus on its applications. Returning to the water-level monitor, the guard $t \in [0, 2]$ in the transitions from 1 to 2 and from 3 to 0, and the invariants in $1, 3$, can be substituted by a parametrized guard $t \in [0, 2+p]$, which states that the latency of the pump is bounded by $2 + p$. Such a parametrized hybrid automaton defines a similarity measure and the model-measuring problem answers the question: what is the maximal latency of the pump tolerated by the system?

However, the model-measuring problem is far more general than parameter synthesis. It can be used to measure the resilience of the model under perturbations. For instance, one can model a pipeline failure scenario, which manifests itself as either water shortage or water absence. In case of water shortage the pump is able to pump only 0.5 water units per second and in water absence 0 units per second. The question "what is the maximal overall time of water shortage tolerated by the water-level monitor" can be expressed as an instance of the model-measuring problem.

Contributions. We continue the line of research set in [11]. We introduce basic notions and notation in Section 2. The remaining part of the paper contains the following contributions:

1. We develop the theory of monotonic parametric hybrid automata, that can be thought of as "weighted" counterparts of hybrid automata (Section 3).
2. We adapt the model-measuring framework [11] to the hybrid automata case (Section 4).
3. We discuss constructions of similarity measures defined by hybrid automata (Section 5).
4. We study which distances on timed words can be expressed by hybrid automata (Section 6).

Related work. Parameter synthesis for hybrid automata, which is a special case of model measuring for hybrid automata, has been extensively studied [7, 8]. However, parameter synthesis is usually considered for arbitrary parameterizations (in particular in [8]), whereas, we are interested only in monotonic parameterizations. The (non-monotonic) parameter synthesis problem becomes quickly undecidable; it is already undecidable for timed automata. Therefore, the methods for (non-monotonic) parameter synthesis are necessarily incomplete or nonterminating.

Monotonic parameterizations of timed automata have been studied in [12]. Our approach is more general. First, it is defined on affine hybrid automata. Second, it allows constraints of the form $t \in [-p, p]$, which are disallowed in [12]. Third, parameter synthesis is a special case of our model-measuring framework.

Evaluation of hybrid systems based on distances has been discussed in [6], where the Skorohod distance has been suggested as the right distance between continuous behaviors. It can be expressed by a hybrid automaton, but it is not known whether that automaton belongs to a class enjoying decidability of the emptiness problem.

2 Preliminaries

A *labeled transition system* is a quadruple (S, Σ, E, s_0), where S is a (finite or infinite) set of states, Σ is an alphabet, E is a relation on $S \times \Sigma \times S$ and s_0 is an initial state. All models considered in this paper are (finite or infinite) transition systems. A word $w = a_1 a_2 \dots$ is a *trace* of a (labeled) transition system M if there is a path $s_0 s_1 \dots$ in M such that for every $i \in [1, |w|]$, $(s_{i-1}, a_i, s_i) \in E$. Such a path is said to be *associated* with w.

2.1 Hybrid automata

We briefly present basic notions regarding hybrid automata and its special case, timed automata. Timed-automata and related topics are discussed in a comprehensive survey [2]. Hybrid automata in general and their connections to timed automata are discussed in [3, 9]. In the following, we consider only automata over finite timed words.

Let X be a set of real-valued variables. An *affine constraint* is a conjunction of terms of the form $s(X) \sim c$, where $c \in \mathbb{Q}$, $\sim \in \{<, \leq, =, \geq, >\}$ and $s(X)$ is a linear combination of the variables from X with rational coefficients. Denote by $\textsc{Aff}(X)$ the set of affine constraints over X. A *valuation* ν is a mapping $\nu : X \mapsto \mathbb{R}$ (written also as $\nu \in \mathbb{R}^X$). A valuation ν satisfies $g \in \textsc{Aff}(X)$, denoted by $\nu \models g$, if the formula $\nu(g)$, obtained from g by instantiating variables from X by their values in ν, is true.

An (affine) *hybrid automaton* $\mathcal{A}$ is a tuple $(\Sigma, L, X, \langle l_0, \nu_0 \rangle, Inv, Flow, \delta, F)$, where (1) Σ is the alphabet of $\mathcal{A}$ (called labels in [9]), (2) L, X are sets of locations and variables, (3) l_0, ν_0 are the initial location and valuation, (4) $Inv : L \mapsto \textsc{Aff}(X)$ defines location invariants, (5) $Flow : L \mapsto \textsc{Aff}(\dot{X})$ defines dynamics of each variable at a given location, where $\dot{x}$ is the derivative of x (also called its *rate*), and $\dot{X} = \{\dot{x} : x \in X\}$ (6) $\delta \subseteq L \times \textsc{Aff}(X \cup X^+) \times (\Sigma \cup \{\epsilon\}) \times L$ is the switch relation, where x^+ denotes the value of x after taking the switch and $X^+ = \{x^+ : x \in X\}$. A switch (l, s, a, l') resets a value of x, if s is consistent with $x \neq x^+$, (7) F is a finite set of acceptance conditions of the form $\langle l, s \rangle$, where $l \in L$ and $s \in \textsc{Aff}(X)$ is a constraint on variables.

Notice that $Flow(l)$ consists of only derivatives and constants, i.e., $\dot{x} \in [0, 2]$ is allowed and $x = \dot{x}$ is not.

We assume that each hybrid automaton has the *time variable* $t \in X$, whose rate is 1 at every location.

We settled for the class of affine hybrid automata, therefore we omit the word "affine".

We define the size of the automaton $\mathcal{A}$, denoted by $|\mathcal{A}|$, as the length of its binary representation, where the constants are represented in binary notation.

An *event* over Σ and X is a pair $\langle a, \nu \rangle$ of a discrete action from $\Sigma \cup \{\epsilon\}$ and a valuation $\nu : X \mapsto \mathbb{R}$. A *timed word* over Σ and X is a finite sequence of events $\langle a_0, \nu_0 \rangle \dots \langle a_k, \nu_k \rangle$ such that the time variable $\nu_0(t), \nu_1(t), \dots$ is a weakly increasing sequence. For a timed word w, we define $untime(w)$ as the projection of w on Σ^*. A *state* $q = \langle l, \nu \rangle$ of $\mathcal{A}$ is a pair of a location $l \in L$ and a valuation ν.

There are two kinds of transitions between states of $\mathcal{A}$: (*i*) *elapse of time*: $\langle l, \nu \rangle \xrightarrow{\tau} \langle l, \nu' \rangle$ iff there is a differentiable function $f : [0, \tau] \mapsto \mathbb{R}^X$, such that $f(0) = \nu$, $f(\tau) = \nu'$ and for all $\lambda \in [0, \tau]$, the valuation $\mu_\lambda : X \cup \dot{X} \mapsto \mathbb{R}$, defined for all $x \in X$ as $\mu_\lambda(x) = f(\lambda)[x]$, $\mu_\lambda(\dot{x}) = (\frac{\partial}{\partial t} f(\lambda))[x]$, satisfies $Flow(l)$, (*ii*) *location switch*: $\langle l, \nu \rangle \underset{a}{\rightarrow} \langle l', \nu' \rangle$ iff there is a switch of $\mathcal{A}$, (l, s, a, l'), such that the valuation $\tau : X \cup X^+ \mapsto \mathbb{R}$, defined for all $x \in X$ as $\tau(x) = \nu(x)$ and $\tau(x^+) = \nu'(x)$, satisfies s. An elapse of time is usually followed by a location switch, thus we define the composition $\xrightarrow{\tau} \circ \underset{a}{\rightarrow}$, denoted by $\underset{a}{\xrightarrow{\tau}}$.

We associate with $\mathcal{A}$ a transition system $\text{Pre}_\mathcal{A} = (L \times \mathbb{R}^X, (\Sigma \cup \{\epsilon\}) \times \mathbb{R}^X, E, \langle l_0, \nu_0 \rangle)$ such that $\langle l_0, \nu_0 \rangle$ is the initial state of $\mathcal{A}$ and $(\langle l, \nu \rangle, \alpha, \langle l', \nu' \rangle) \in E$ if $\alpha \in \Sigma \cup \{\epsilon\}$ and there is $\tau > 0$ such that $\mathcal{A}$ has a transition $\langle l, \nu \rangle \underset{\alpha}{\xrightarrow{\tau}} \langle l', \nu' \rangle$. Paths in $\text{Pre}_\mathcal{A}$ are called *runs* of $\mathcal{A}$.

A state $\langle l, \nu \rangle$ of $\text{Pre}_\mathcal{A}$ is *accepting* iff there is a constraint $(l', s) \in F$ from $\mathcal{A}$ such that $l = l'$ and ν satisfies s.

For a hybrid automaton $\mathcal{A}$, we define $[\mathcal{A}]$ as the set of traces from $[\text{Pre}_\mathcal{A}]$ whose last state is accepting. A timed word w is accepted by $\mathcal{A}$ iff there is a trace $v \in [\mathcal{A}]$ whose projection to $\Sigma \times \mathbb{R}^X$ is w. We will write $\mathcal{A}_M$ to indicate that $\mathcal{A}_M$ generates a transition system M such that $[M] = [\mathcal{A}_M]$.

The *emptiness problem* for hybrid automata asks, given hybrid automaton $\mathcal{A}$, is $[\mathcal{A}]$ nonempty? That problem is also referred to as the *reachability* problem as it is equivalent to reachability of an accepting state in $\text{Pre}_\mathcal{A}$.

2.2 Product of hybrid automata

Let $\mathcal{A}_i = (\Sigma, L_i, X_i, \langle l_{0,i}, \nu_{0,i} \rangle, Inv_i, Flow_i, \delta_i, F_i)$, for $i \in \{1, 2\}$, be hybrid automata over Σ. We define *the product* of $\mathcal{A}_1$ and $\mathcal{A}_2$, denoted by $\mathcal{A}_1 \times \mathcal{A}_2$, as the hybrid automaton $\mathcal{A}_1 \times \mathcal{A}_2 = (\Sigma, L, X, \langle l_0, \nu_0 \rangle, Inv, Flow, \delta, F)$ such that:

(i) $L = L_1 \times L_2$ and $X = X_1 \cup X_2$,

(ii) $l_0 = \langle l_{0,1}, l_{0,2} \rangle$, $\nu_0 = \nu_{0,1} \cup \nu_{0,2}$, and

for all $l_1 \in L_1, l_2 \in L_2$:

(iv) $Inv(\langle l_1, l_2\rangle) \equiv Inv_1(l_1) \wedge Inv_2(l_2)$

(v) $Flow(\langle l_1, l_2\rangle) \equiv Flow_1(l_1) \wedge Flow_2(l_2)$

(vi) $\delta = \{(\langle l_1, l_2\rangle, s_1 \wedge s_2, a, \langle l_1', l_2'\rangle) : (l_1, s_1, a, l_1') \in \delta_1,\ (l_2, s_2, a, l_2') \in \delta_2\}$

(vii) $F = \{(\langle l_1, l_2\rangle, s_1 \wedge s_2) : (l_1, s_1) \in F_1, (l_2, s_2) \in F_2\}$

The automaton $\mathcal{A}_1 \times \mathcal{A}_2$ recognizes the intersection of languages recognized by $\mathcal{A}_1$ and $\mathcal{A}_2$. (This holds only for affine hybrid automata, as they have piecewise-linear trajectories.)

2.3 Rectangular hybrid automata

An affine constraint is *rectangular* iff it is a finite conjunction of expressions $x \sim c$, where $c \in \mathbb{Q}$ and $\sim \in \{<, \leq, =, \geq, >\}$. A hybrid automaton is *rectangular* iff all constraints, are rectangular or of the form $x = x^+$ (which occur only in switches).

A rectangular hybrid automaton is *initialized* iff for each switch (l_1, s, a, l_2) and every $x \in X$, if the flow of x changes, i.e, $Flow(l_1) \restriction_x \neq Flow(l_2) \restriction_x$, then x is reset at that switch. E.g. every timed automaton is an initialized rectangular automaton [10].

Theorem 1 ([10]) *The emptiness problem for initialized rectangular automata is PSPACE-complete.*

A subset of $\mathbb{R}^n$ is *compact* iff it is bounded and closed. A constraint is *compact* iff it defines a compact subset of $\mathbb{R}^n$. Finally, a hybrid automaton is *compact* iff all constraints defining its invariants, flows, the transition relation and the acceptance conditions are compact.

2.4 Parametric hybrid automata

A variable x is a *parameter* in a hybrid automaton if its rate at each location is 0 and in every switch we have the constraint $x = x^+$. A *parametric hybrid automaton* $\mathcal{A}$ is a hybrid automaton with distinguished subsets of variables that are parameters. We write $\mathcal{A}[\vec{p}]$ to indicate that $\mathcal{A}$ has parameters $\vec{p}$. Since parameters values do not change during a run, they can be instantiated to (rational) constants.

We say that a parametric hybrid automaton belongs to a class $\mathcal{C}$ iff for every instantiation of parameters, the resulting automaton belongs to $\mathcal{C}$. E.g. even though a parametric hybrid automaton $\mathcal{A}$ has a non-rectangular constraint $x \leq p$, each instantiation of p makes $x \leq p$ rectangular and we consider such an automaton to be rectangular.

The emptiness problem has its parametric counterpart. *The parametric emptiness problem* asks, given a parametric hybrid automaton, is there an instantiation of its parameters (by rational numbers) such that the resulting automaton recognizes a nonempty language.

Observe that parameters in a hybrid automaton are instantiated in ν_0, whereas in the parametric emptiness problem they can be arbitrarily instantiated. It follows that the emptiness problem implies the parametric emptiness problem, but not vice versa.

2.5 Weighted timed automata

A *weighted timed automaton* is a timed automaton augmented by weights associated with its locations and switches, $C : L \cup \delta \mapsto \mathbb{N}$. The value of a run $\langle l_0, \nu_0\rangle \xrightarrow[a_1]{\tau_1} \langle l_1, \nu_1\rangle \xrightarrow[a_2]{\tau_2} \ldots \xrightarrow[a_k]{\tau_k} \langle l_k, \nu_k\rangle$ is given by $\sum_{i=0}^{k-1} C(l_i)\tau_{i+1} + \sum_{i=0}^{k-1} C(e_i)$, where e_i is the switch taken in the transition $\langle l_i, \nu_i\rangle \xrightarrow[a_{i+1}]{\tau_{i+1}} \langle l_{i+1}, \nu_{i+1}\rangle$. The value of a timed word w assigned by a weighted timed automaton $\mathcal{A}$, denoted by $L_{\mathcal{A}}(w)$, is the infimum of the set of values of all accepting runs of $\mathcal{A}$ on w. Timed words that are rejected by $\mathcal{A}$ have infinite value.

The emptiness question for non-weighted automata has the following weighted counterpart:

Definition 2 *The* optimal-value question *asks, given a weighted timed automaton $\mathcal{A}$, to compute the infimum of $L_{\mathcal{A}}(w)$ over all timed words.*

It has been shown in [4] that:

Theorem 3 *The optimal-value question for timed automata can be computed in polynomial space.*

Weighted timed automata defined as above are referred to in the literature as *linearly-priced timed automata*. Observe that weighted timed automata are a special kind of parametric affine hybrid automata. Indeed, a timed part of a timed automaton is translated directly to an affine hybrid automaton and the value is represented by a *value variable* val. For a switch e, val is updated according to $\text{val}^+ = \text{val} + C(e)$. In a location l, the rate of val is equal $C(l)$. The value of val does not appear in any constraints except for acceptance conditions, where val is compared with a threshold parameter tr, where $\text{val} \leq \text{tr}$ states that we are interested in runs whose values are bounded by tr.

3 Parametric hybrid automata

In this section we develop a unified theory of "weighted" hybrid automata over finite timed words, which is applied in Section 4. Richness of hybrid automata allows encoding value directly as a continuous variable and avoiding auxiliary components as in discrete weighted automata. However, that richness raises decidability issues.

However, the emptiness problem for the whole class of (affine) hybrid automata is undecidable. Thus, we shall restrict ourselves to subclasses of hybrid automata that have the emptiness and parametric emptiness problems decidable (weighted timed automata, initialized rectangular automata). The following definitions help us to address decidability issues.

Definition 4 *A class of hybrid automata $\mathfrak{C}$ is: (1)* weakly decidable *iff the membership (of an automaton to $\mathfrak{C}$) and the emptiness problems are decidable over $\mathfrak{C}$ (2)* strongly decidable *iff the membership, the emptiness and parametric emptiness problems are decidable over $\mathfrak{C}$.*

The class of parametric timed automata is weakly decidable but not strongly decidable. This implies that even if there is a (rational) instantiation of parameters such that the resulting automaton recognizes a nonempty language, there is no bound on the length of binary representation rational parameters. If there was a bound, there would be finitely many instances of the automaton preserving the bound, and one could just test for emptiness all of those instances. Thus, instead of computing exact values of parameters, we have to settle for approximation.

We define the class of admissible parameters for $\mathcal{A}$, denoted by $\mathfrak{Par}(\mathcal{A})$, as $\{\vec{p} : [\mathcal{A}[\vec{p}]] \neq \emptyset\}$. We are interested in approximation of "minimal" elements of $\mathfrak{Par}(\mathcal{A})$ (which is called the *quantitative emptiness problem* or the *optimal-value problem* in the weighted automata case.) Approximation and minimality are defined by a vector norm $|\cdot|$. For a given norm and $\epsilon > 0$, we say that $\vec{q}$ is an ϵ-approximation of $\vec{p}$ iff $|\vec{q} - \vec{p}| \leq \epsilon$. A vector $\vec{p}$ is minimal in a set A if $\vec{p} \in A$, and for every $\vec{q} \in \mathcal{A}$, $|\vec{p}| \leq |\vec{q}|$. As $\mathfrak{Par}(\mathcal{A})$ may not have minimal elements, we consider its closure in $\mathbb{R}^n$, $cl(\mathfrak{Par}(\mathcal{A}))$. Each coordinate of $\mathfrak{Par}(\mathcal{A})$ is positive, hence $cl(\mathfrak{Par}(\mathcal{A}))$ contains minimal elements.

Definition 5 *A class of hybrid automata $\mathfrak{C}$ admits* parameter approximation *iff there is a procedure that for a given $n > 0$ and $\mathcal{A} \in \mathfrak{C}$, returns a 2^{-n}-approximation of a minimal element of $cl(\mathfrak{Par}(\mathcal{A}))$ if $\mathfrak{Par}(\mathcal{A})$ is nonempty and $\emptyset$ otherwise.*

In the rest of this section we give sufficient conditions for a class $\mathfrak{C}$ to admit effective parameter approximation. The complexity of parameter approximation is given separately w.r.t. the size of an automaton $|\mathcal{A}|$ and a precision n.

Assume that a class of hybrid automata $\mathfrak{C}$ is weakly decidable, but not strongly decidable. There are two possible reasons for the undecidability: density and unboundedness of the domain of possible parameters, the rational numbers, which corresponds to infinite domain of denominators (density) and numerators (unboundedness). The undecidability of the parametric emptiness problem for timed automata is caused by the density. Indeed, even a restricted problem, are there parameters from the interval $[1, 2]$ such that $[\mathcal{A}[\vec{p}]]$ is nonempty, is undecidable [13]. Density, as a source of undecidability, can be eliminated by restricting parameterizations to be monotonic:

Definition 6 *We say that a parametric hybrid automaton $\mathcal{A}$ is* monotonic *iff for all $\vec{p}, \vec{q} \in \mathbb{Q}^k$, $\vec{p} \leq \vec{q}$ ($\vec{q}$ is greater than $\vec{q}$ component-wise) implies $[\mathcal{A}[\vec{p}]] \subseteq [\mathcal{A}[\vec{q}]]$*

There are simple sufficient conditions that guarantee monotonicity of a given hybrid automaton. E.g. an automaton with constraints $x - p \leq 0$ or $x + p \geq 0$ is monotonic w.r.t. p as higher values of p lead to weakening the constraints and admittance of more runs. Still, given a hybrid automaton, even a timed automaton, it is not possible to decide whether it is monotonic.

Proposition 7 *Monotonicity of parametric timed automata is undecidable.*

All monotonic timed automata are strongly decidable, but in the rich class of hybrid automata, monotonicity alone does not make a weakly decidable class strongly decidable. The unboundedness of the domain of parameters may cause undecidability of the parametric emptiness problem even with monotonic parametrizations. Indeed, consider the class stopwatch automata over bounded time, i.e., parametric stopwatch automata that have a single parameter bounding the duration of each run. Such automata are monotonic and their class is weakly decidable [5]. However, that class is not strongly decidable as its strong decidability is equivalent to weak decidability of the class of stopwatch automata.

In order to eliminate unboundedness as a source of undecidability, we define a small parameter property called *f-boundedness*.

Definition 8 *Let f be a computable function. A class of hybrid automata $\mathfrak{C}$ is* f-bounded *iff for every $\mathcal{A} \in \mathfrak{C}$, if there is a tuple $\vec{p}$ such that $[\mathcal{A}[\vec{p}]] \neq \emptyset$, then there a tuple $\vec{q}$ whose components are from $[0, f(|\mathcal{A}|)]$ such that $[\mathcal{A}[\vec{q}]] \neq \emptyset$.*

An f-bounded class of parametric hybrid automata that are monotonic admits effective parameter approximation procedure. Basically, for a given $\mathcal{A} \in \mathfrak{C}$ and $\epsilon = 2^{-n}$, one can do binary search for parameters with the minimal norm, where each parameter is from the set $\{\frac{i}{j} : j \in \{1, \ldots, 2^n\}, i \in \{0, \ldots, j \cdot f(|\mathcal{A}|)\}\}$. Observe that the length of instantiated parameters in the fixed-point representation is bounded by $log(f(|\mathcal{A}|)) + n$. It follows that each automaton that results from such an instantiation of $\mathcal{A}$ has the size bounded by $\mathcal{A} + PM(\mathcal{A})(log(f(|\mathcal{A}|)) + n)$, where $PM(\mathcal{A})$ is the number of instances of parameters in $\mathcal{A}$.

Proposition 9 *Every weakly decidable and f-bounded class of monotonic hybrid automata $\mathfrak{C}$ admits parameter approximation in time $O((f(|\mathcal{A}|) + n) \cdot T(|\mathcal{A}| + PM(\mathcal{A}) \cdot (log(f(|\mathcal{A}|)) + n)))$, where $T(\cdot)$ is the complexity of the emptiness problem for $\mathfrak{C}$.*

f-boundedness of a class of weakly decidable hybrid automata, which are monotonic, is equivalent to strong decidability in the following sense.

Proposition 10 *Let $\mathfrak{C}$ be a weakly decidable class of monotonic hybrid automata. The class $\mathfrak{C}$ is strongly decidable iff there is a computable function f such that $\mathfrak{C}$ is f-bounded.*

The classes of monotonic weighted timed automata and compact initialized rectangular automata are exponentially bounded classes. Parameters in (affine) hybrid automata occur only in affine terms. Thus, to show that a class $\mathfrak{C}$ is exponentially bounded, it is sufficient to show that if an automaton from $\mathfrak{C}$ has an accepting run, it has an accepting run in which all variables are exponentially bounded.

A straightforward analysis of the region graph of a timed automaton yields that if a timed automaton has an accepting run, then there is a timed word $(a_0, t_0) \ldots (a_k, t_k)$ accepted by this timed automaton with k and $t_k - t_0$ exponentially bounded in the size of the automaton. It follows that each timed automaton that recognizes a nonempty language has a run in which values of all clocks and the value are exponentially bounded.

Proposition 11 *The class of monotonic weighted timed automata (over finite words) is exponentially bounded.*

Compact initialized rectangular automata have bounded rates of their variables and they recognize precisely timed languages recognized by timed automata [10]. Thus, if a compact initialized rectangular automaton has an accepting run, it accepts a timed word $(a_0, t_0) \dots (a_k, t_k)$ such that k, $t_k - t_0$ are exponentially bounded in the size of the automaton. It follows, that it has an accepting run in which the values of all variables are exponentially bounded. Hence, we have:

Proposition 12 *The class of monotonic compact initialized rectangular automata (over finite words) is exponentially bounded.*

In consequence, we have:

Corollary 13 *Let $|\cdot|$ be a vector norm computable in linear time. The class of monotonic compact initialized rectangular automata admits parameter approximation in polynomial space w.r.t. the automaton size and linear time w.r.t. a given precision n.*

Finally, the optimal parameters of weighted timed automata can be actually computed as they are always integers or arbitrarily close to integers [4], which makes their infima integers. Thus, it is sufficient to approximate the optimal parameters only up to ϵ, such that each ϵ-neighborhood contains at most one vector of integers.

Proposition 14 *Given a monotonic weighted timed automaton $\mathcal{A}$, the value $\inf\{|\vec{p}| : \vec{p} \in \mathfrak{Par}(\mathcal{A})\}$ can be computed in polynomial space in $|\mathcal{A}|$.*

Recall that $\inf\{|\vec{p}| : \vec{p} \in \mathfrak{Par}(\mathcal{A})\} = \infty$ if $\mathfrak{Par}(\mathcal{A})$ is empty.

4 Model-measuring framework

The model-measuring problem [11] asks, given a model M and a specification φ, what is the maximal distance ρ such that all models M' within that distance from M satisfy (or violate) φ. In this section, we adapt the model-measuring problem to the hybrid automata setting. We begin with basic definitions from [11].

Let M be a transition system (model). A *similarity measure* (of M) is a function d_M from timed words into positive real numbers such that for all traces w of M, $d_M(w) = 0$. Such a similarity measure extends to transition systems by $d_M(M') = \sup\{d_M(w) : w \text{ is a trace of } M'\}$. Every similarity measure results from a distance function via fixing the first argument.

Definition 15 *Let M be a transition system and d_M be a similarity measure. For a specification P, the* stability radius *of P in M (w.r.t. d_M), denoted by $sr_{d_M}(P)$, is defined as follows:*
(1) if $M \models P$, $sr_{d_M}(P) = \sup\{\rho \geq 0 : \forall M'(d_M(M') < \ \rho \Rightarrow M' \models P)\}$,
(2) if $M \models \neg P$, $sr_{d_M}(P) = sr_{d_M}(\neg P)$,
(3) otherwise, $sr_{d_M}(P) = 0$.

Now, these definitions are specialized to the hybrid automata case. We begin with representing similarity measures by monotonic hybrid automata, a hybrid counterpart of weighted automata.

Definition 16 *Let $|\cdot|$ be a norm computable in linear time. A (hybrid) similarity measure d_M is* automatic *iff there is a monotonic hybrid automaton $\mathcal{A}_{dist}$ such that $d_M(M') = \inf\{r : \exists \vec{p}.|\vec{p}| \leq r \wedge [M'] \subseteq [\mathcal{A}_{dist}][\vec{p}]\}$.*

We assume that a specification is given by a hybrid automaton $\mathcal{A}_P$ that accepts all timed words that violate the specification. We call specifications given in such a way *automatic*. Observe that a transition system M, given by a hybrid automaton $\mathcal{A}_M$, satisfies an automatic property P ($M \models P$) iff the languages recognized by $\mathcal{A}_M$ and $\mathcal{A}_P$ are disjoint. Thus, model checking for hybrid automata reduces to the emptiness problem for hybrid automata.

We define the *model measure* on the basis of the stability radius by scaling the value the stability radius from $[0, \infty]$ to $[\frac{1}{2}, 1]$ if $M \models P$, and $[0, \frac{1}{2}]$ otherwise.

Definition 17 *The* model-measuring problem *is defined as follows: given an automatic similarity measure d_M and an automatic specification P, compute $[P]_{d_M}$ defined by*

(i) if $M \models P$, $[P]_{d_M} = 1 - 2^{-sr_{d_M}(M,P)-1}$ $(\in [\frac{1}{2}, 1])$,

(ii) if $M \models \neg P$, $[P]_{d_M} = 1 - [\neg P]_{d_M}$ $(\in [0, \frac{1}{2}])$,

(iii) otherwise, $[P]_{d_M} = \frac{1}{2}$.

Observe that the minimal norm of parameters $\vec{p}$ such that $[\mathcal{A}_{dist}[\vec{p}]]$ and $[\mathcal{A}_P]$ have nonempty intersection is exactly the stability radius of P in M w.r.t. d_M. Thus, the model-measuring problem for hybrid specifications reduces to the problem of finding the minimal norm of parameters such that $(\mathcal{A}_{dist} \times \mathcal{A}_P)[\vec{p}]$, the product of automata $\mathcal{A}_{dist}$ and $\mathcal{A}_P$, recognizes a nonempty language.

We can restrict instances of the model-measuring problem (a similarity measure and a specification) to be represented by hybrid automata from a given class. We call such a restricted problem *the model-measuring problem over* $\mathfrak{C}$.

As noted in Section 3 the minimal norm of a vector of parameters can be a rational number with arbitrarily long representation. For that reason, we settle for approximation. Corollary 13 and Proposition 14 imply the following:

Theorem 18 *The model measuring problem over compact initialized rectangular automata can be* ***approximated*** *in polynomial space in the size of automata representing d_M and P.*

The model measuring problem over weighted timed automata can be ***computed*** *in polynomial space in the size of automata representing d_M and P.*

5 Modeling hybrid similarity measures

In this section we present a systematic approach to the construction of automatic hybrid similarity measures. The main difficulty originates from the following issues. First, the model M is usually complex, therefore modifying its internal structure is a complicated and error-prone task. Ideally, the construction method would yield a similarity measure given only the original hybrid automaton and a description of allowed perturbations. Second, the outcome of the construction should be a parametric hybrid automaton that admits effective approximation of minimal parameters for which its intersection with another hybrid automaton is nonempty. The presented approach addresses both issues.

Hybrid automata combine discrete and continuous control via extending finite state automata by continuous variables. Thus, for modeling automatic hybrid similarity measures, we extend the (discrete) hypervisor approach [11] to the hybrid case. The main idea behind the hypervisor approach is to introduce an external component, called the *hypervisor*, whose task is to govern the execution of the original automaton $\mathcal{A}_M$ by providing alternative switch relations, invariants and flows. A hypervisor runs an external hybrid automaton that using its current location selects the current switch relation, invariants and flows.

Composition of $\mathcal{A}_M$ with a hypervisor produces a monotonic hybrid automaton. In order to guarantee monotonicity of the resulting automaton, we require all constraints to be monotonic in the following sense. An affine constraint $t[\vec{p}]$ parametrized by $\vec{p}$ is monotonic iff for all vectors of real numbers $\vec{c_1}, \vec{c_2}$, if $\vec{c_1} \leq \vec{c_2}$, then $t[\vec{c_1}]$ implies $t[\vec{c_2}]$.

Definition 19 *A* hypervisor *for a hybrid automaton $\mathcal{A}_M = (\Sigma, L_M, X_M, \langle l_{0,M}, \nu_{0,M}\rangle, Inv_M, Flow_M, \delta_M, F_M)$ is a quadruple $H = (\mathcal{A}_H, \tau_H, \mathfrak{Inv}_H, \mathfrak{Flow}_H)$ satisfying:*

(i) $\mathcal{A}_H = (\Sigma, L_H, \langle l_{0,H}, \nu_{0,H}\rangle, X_H, Inv_H, Flow_H, \delta_H, F_H)$ is a parametric hybrid automaton.

(ii) for every $x \in X_M \cap X_H$, $\nu_{0,M}(x) = \nu_{0,H}(x)$

(iii) $\tau_H : L_H \mapsto \mathcal{P}(L_M \times \textsc{Aff}(X_M \cup X_M^+) \times \Sigma \times L_M)$, where $\mathcal{P}(A)$ denotes the power set of A,

(iv) $\mathfrak{Inv}_H : L_H \mapsto (L_M \mapsto \textsc{Aff}(X_M))$,

(v) $\mathfrak{Flow}_H : L_H \mapsto (L_M \mapsto \textsc{Aff}(\dot{X_M}))$,

(vi) $\mathcal{A}_H$ *has the initial state* $\langle l_I, \nu \rangle$, *an* idle state, *such that* $\tau_H[l_I] = \delta_M$, $\mathfrak{Inv}_H[l_I] = Inv_M$, $\mathfrak{Flow}_H[l_I] = Flow_M$, *and for all* $a \in \Sigma$, $\mathcal{A}_H$ *has a switch* $(l_I, \top, a, l_I)$, *where* $\top$ *is the empty constraint that is always satisfied,*

(vii) all affine constraints in $\tau_H, \mathfrak{Inv}_H, \mathfrak{Flow}_H$ *and* $\mathcal{A}_H$ *are monotonic.*

At each step, the functions $\tau_H, \mathfrak{Inv}_H, \mathfrak{Flow}_H$ determine the transition relation, invariants and flows for $\mathcal{A}_M$. Intuitively, they should encode modifications applied to these components of $\mathcal{A}_M$ rather than their complete descriptions. For example:
(blind a-transitions) Consider $l_a \in L_H$ such that $\tau_H[l_a] = \{(l, s, b, l') : (l, s, a, l') \in \delta_M, b \in \Sigma\}$, i.e., the automaton moves as it would have read an event with a. $\tau_H[l_a]$ can be simply defined uniformly on δ_M, i.e., regardless of the complexity of δ_M.
(bounded deviation of x) Consider $\tau_H[l_x] = \{(l, s, a, l') : (l, s', a, l') \in \delta_M, s' \equiv s[x/(x+y)] \wedge y \leq q \wedge y \geq -q\}$, where $s[x/(x+y)]$ results from substitution x by $x+y$ in s. Intuitively, we allow values of x to deviate from its intended values by at most q.

For a hybrid automaton $\mathcal{A}_M$ and a hypervisor H as in Definition 19, we define the semi-direct product of $\mathcal{A}_M$ and H. The *semi-direct product* $\mathcal{A}_M \ltimes H$ is a parametric hybrid automaton $(\Sigma, L, X, \langle l_0, \nu_0 \rangle, Inv, Flow, \delta, F)$ such that:

- $L = L_M \times L_H$, and $X = X_M \cup X_H$,
- $l_0 = \langle l_{0,M}, l_{0,H} \rangle$, $\nu_0 = \nu_{0,M} \cup \nu_{0,H}$
- $Inv(\langle l_M, l_H \rangle) \equiv \mathfrak{Inv}_H[l_H](l_M) \wedge Inv_H(l_H)$
- $Flow(\langle l_M, l_H \rangle) \equiv \mathfrak{Flow}_H[l_H](l_M) \wedge Flow_H(l_H)$
- $(\langle l_M, l_H \rangle, s, a, \langle l'_M, l'_H \rangle) \in \delta$ iff
  - (i) $(l_H, s_H, a, l'_H) \in \delta_H$, $(l_M, s_M, a, l'_M) \in \tau_H[l'_H]$, and $s = s_M \wedge s_H$, or
  - (ii) $a = \epsilon$, $(l_H, s_H, \epsilon, l'_H) \in \delta_H$ and $l_M = l'_M$
- $F = \{\langle (l_M, l_H), s_M \wedge s_H \rangle : \langle l_H, s_H \rangle \in F_H, \langle l_M, s_M \rangle \in F_M\}$

The semi-direct product defines an automatic hybrid similarity measure. Indeed, due to existence of the idle location, regardless of the values of the parameters, each timed word accepted by $\mathcal{A}_M$ is also accepted by $\mathcal{A}_M \ltimes H$. More precisely, as $\mathcal{A}_M$ can be a parametric automaton itself, for every $\vec{p} \in \mathbb{R}^m$, $[\mathcal{A}_M[\vec{p}]] \subseteq [(\mathcal{A}_M \ltimes H)[\vec{p}]]$. Conversely, any automatic hybrid similarity measure can be obtained by the hypervisor construction. Indeed, for any automatic hybrid similarity measure defined by $\mathcal{A}$ one can define a hypervisor H, which neglects $\mathcal{A}_M$ and simulates $\mathcal{A}$. Therefore, the hypervisor approach is complete, i.e., every automatic hybrid similarity measure can be obtained as a semi-direct product.

All constraints introduced by the hypervisor are monotonic, therefore if $\mathcal{A}_M$ is monotonic, $\mathcal{A}_M \ltimes H$ is monotonic as well. In contrast, there is no natural condition on the hypervisor that would guarantee that the semi-direct product of an automaton from a given class $\mathfrak{C}$ and that hypervisor belongs to $\mathfrak{C}$. The reason for that is that those conditions heavily depend of the class $\mathfrak{C}$.

Definition 20 *Let* $\mathfrak{C}$ *be a class of hybrid automata. Let* $\mathcal{A} \in \mathfrak{C}$ *and let* H *be a hypervisor* H *for* $\mathcal{A}$. *The hypervisor* H *is* admissible *for* $\mathfrak{C}$ *with* $\mathcal{A}$ *iff* $\mathcal{A} \ltimes H$ *belongs to* $\mathfrak{C}$.

The following facts can be easily verified by the reader:

Fact 21 *Every hypervisor* $H = (\mathcal{A}_H, \tau_H, \mathfrak{Inv}_H, \mathfrak{Flow}_H)$, *that satisfies the following conditions is admissible for the class of weighted timed automata (regardless of* $\mathcal{A}_M$*): (1)* $\mathcal{A}_H$ *is a (linearly) weighted timed automaton, (2)* $\mathfrak{Inv}_H$ *is rectangular, and for every* $l_h \in L_H$, *for every variable* x *except for the value variable wt: (3) every switch in* $\tau_H[l_h]$ *preserves the value of* x *or resets it to* 0, *(4) for all* $l_m \in L_M$, $\mathfrak{Flow}_H[l_H](l_m)$ *implies that all slopes of all variables equal* 1.

Fact 22 *Let $\mathcal{A}_M$ be a compact initialized rectangular automaton. Every hypervisor H such that all components are compact and rectangular, and $\mathcal{A}_M \ltimes H$ is initialized, is admissible for the class of compact initialized rectangular automata with $\mathcal{A}_M$.*

Example 23 ***(Pacemaker)*** *Consider a model of a pacemaker, which consists of three components: the sensor, controller and electrodes. The goal of the pacemaker is to ensure that the pulse is in the range 60-70 beats per minute. If there is no heartbeat for a predefined time, the controller sends a signal to the electrodes, which in turn charge their capacitors for some predefined time and fire an impulse.*

The following questions can be expressed (and then approximated) as instances of the model-measuring problem: what are the maximal impulse firing time and latency of the sensor such that the pacemaker model still meets the specification.

The impulse firing time is modeled by a clock x with a predefined time limit T_c. Basically, we need to change this predefined time to a parameter p and ask, what is the maximum value of p, such that the model still meets the specification. This is expressed by a simple hypervisor that has only two locations, an idle location and a parametric location q_p. All the constraints in $\tau_H[q_p], \mathfrak{Inv}_H[q_p], \mathfrak{Flow}_H[q_p]$ remain the same as in $\delta_M, Inv_M, Flow_M$, except that T_c is substituted by $T_c + p$.

The latency of the sensor can be modeled by a hypervisor that captures every heartbeat event and forwards it to the model with a delay. The hypervisor H is defined as $(\mathcal{A}_H, \tau_H, \mathfrak{Inv}_H, \mathfrak{Flow}_H)$, where $\mathcal{A}_H$ is a parametrized timed automaton, which has four locations: an idle location l_I, waiting for a heartbeat l_w, delaying the delivery l_d and delivering a heartbeat to the model l_r. The automaton $\mathcal{A}_H$ can be in the location l_d not longer than p, which is a parameter restricting sensor delays and it can be in the location l_r only for 0 time units. Both $\mathfrak{Inv}_H$ and $\mathfrak{Flow}_H$ in l_w, l_d, l_r are as is the original model. Then, $\tau_H[l_w]$ is equal to δ_M, which corresponds to the usual behavior of the model. The transition relation $\tau_H[l_d]$ specifies that the hypervised automaton ignores a heartbeat event. Finally, $\tau_H[l_r]$ is a blind a transition as defined above; it specifies that the hypervised automaton acts as it would have got a heartbeat event.

Observe that in the above cases the model is a timed automaton and the hypervisors are parametrized timed automata. Therefore, the maximal firing time and the maximal sensor latency can be computed (not only approximated) in polynomial space (Theorem 18).

The impulse firing time and the latency of the sensor from Example 23 represent two types of similarity measures. The similarity measure related to the impulse firing time strongly depends on the model as it essentially asks what is the maximal value of a bound in a specific constraint, whereas the one related to the latency can be described independently of the model. We refer to those two types as *model-dependent* and *model-independent*.

Model-dependent similarity measures are more expressible as any model-independent similarity measure can be considered as model-dependent. On the other hand, tweaking a model may lead to unpredictable results (Example 24). We discuss both approaches in the following sections.

5.1 Model-dependent similarity measures

We begin with an example showing that parameterizing models can effect in non-obvious faults.

Example 24 ***(Scheduler)*** *Consider a system consisting of a scheduler S and processes $P_1, \ldots, P_k$. Each process has two private clocks t_I, t_T, clock of individual instructionss and a total time clock. The clock t_I is used to model duration of execution of an instruction. Each instruction has its own duration and the clock t_I is reset after each execution of an instruction. The clock t_T is used for preemptive scheduling. A process P can execute instructions only when t_T is below the length of a time slice for P. This clock can be reset by the scheduler; processes are scheduled by resetting their clocks. The scheduler has its own clock that is used to determine times of context switches. An example of a process P_1 with the time slice 10 is depicted in Figure 3. Suppose the length of time slice granted to P_1 is parametrized by p, and the tested value of the parameter would be 7. Then, after instructions* INC(X), DEC(X)*, the process P_1 is about to execute $dec(x)$ and $t_T = 5$. Then, after 2 time units another process is scheduled, but P_1 executes* DEC(X) *which takes 3 time units before it is suspended. Indeed, when P_1 is resumed*

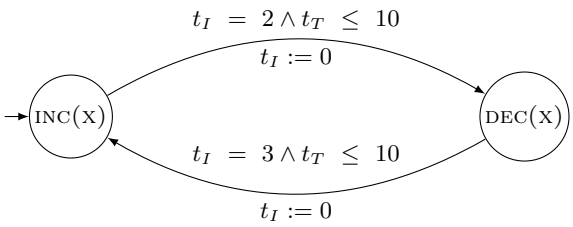


Figure 3: An automaton modeling the process P_1.

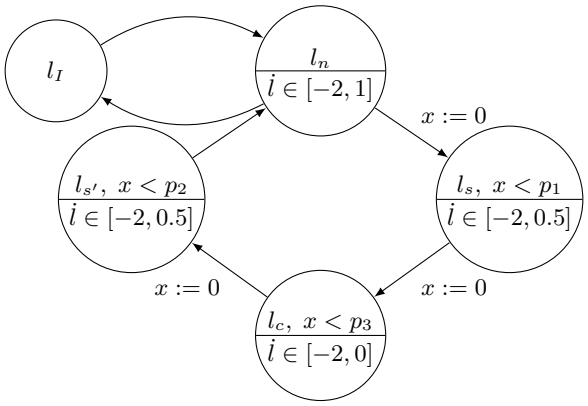


Figure 4: $\mathcal{A}_H$ in the hypervisor modeling the pipeline failure.

it immediately finishes the execution of DEC(X). *This means that for* 1 *time unit two processes were running in parallel, which cannot happen in a single-processor model. The problem did not manifest itself in the original model because the length of time slices is equal to the sum of executing times of instructions. That problem can be solved by compelling the scheduler to reset both clocks t_T and T_I, which corresponds to discarding partial computations.*

Another example is a modification of a water-level monitor [1] presented in the introduction.

Example 25 ***(Water-level monitor)*** *Both similarity measures discussed in the introduction, the one that parametrizes the latency of the pump and the one modeling the water pipeline failure, can be expressed with the hypervisor approach. The hypervisor that parametrizes the latency of the pump is virtually the same as the hypervisor parameterizing the impulse firing time in Example 23; it substitutes the constant* 2 *by* $2 + p$ *in all constraints.*

The hypervisor modeling the water pipeline failure scenario consists of a hybrid automaton $\mathcal{A}_H$ that has 5 *locations: an idle location l_I, l_n corresponding to normal water supply, $l_s, l_{s'}$ corresponding to water shortage and l_c corresponding to water absence. It is depicted in Figure 4.*

Moreover, $\mathcal{A}_H$ has a clock x and the parameters that bound the periods of the water shortage before and after the water absence (p_1, p_3) and the water absence period (p_3). The constraints on the bottom of each node represent additional constraints on the flow of the original model.

Finally, the norm $2 \cdot (|p_1| + |p_3|) + |p_2|$ applied to the optimal values of parameters corresponds to the maximal overall pipeline failure time tolerated by the water-level monitor.

5.2 Model-independent similarity measures

In this section we show how to define model-independent similarity measures for hybrid automata. A natural way to do that is to define a "distance" between timed words, and on its basis define a similarity measure as a "distance" of timed words from the set of traces of the model L_M. Similarity measures defined in that way are clearly model-independent.

In the following, we discuss "distances" between timed words that are computed by monotonic hybrid automata. We call such "distances" *hybrid-automatic weighted relations*, as we do require them to be symmetric and satisfy the triangle inequality. Also, they generalize automatic relations in two ways: their range is the real numbers instead of Boolean values and they are computed by hybrid automata.

We begin with preliminary definitions. Let w_1, w_2 be timed words over Σ and X. For $i \in \{1,2\}$, we define the labeling σ_i on events as $\sigma_i(\langle a, \nu\rangle) = \langle (a,i), \nu_i\rangle$, where $\nu_i : (X \times \{i\}) \cup \{t\} \mapsto \mathbb{R}$ is defined as $\nu_i(\langle x, i\rangle) = \nu(x)$ and $\nu_i(t) = \nu(t)$. The (event) labellings σ_1, σ_2 extend to timed words by applying themselves to each event. We define *the disjoint union* of w_1 and w_2, denoted by $w_1 \oplus w_2$, as the timed word $\Sigma \times \{1,2\}$ consisting of the union of events of timed words $\sigma_1(w_1)$ and $\sigma_2(w_2)$, E.g. $\langle a, 0.4\rangle\langle b, 2.1\rangle \oplus \langle b, 0.3\rangle\langle b, 0.4\rangle = \langle (b,2), 0.3\rangle\langle (a,1), 0.4\rangle\langle (b,2), 0.4\rangle\langle (b,1), 2.1\rangle$.

A *weighted relation* is a generalization of the usual relation by allowing its characteristic function to range over $\mathbb{R}^+ \cup \{\infty\}$. We say that a (binary) weighted relation on timed words R is a *hybrid-automatic weighted relation* iff there is a monotonic hybrid automaton $\mathcal{A}_R$ such that for all timed words v, w we have $vRw = \inf\{|\vec{p}| : \mathcal{A}_R[\vec{p}] \text{ accepts } v \oplus w\}$. In particular, vRw may be equal to the value of $v \oplus w$ assigned by a weighted timed automaton $\mathcal{A}_R$. Indeed, it suffices for $\mathcal{A}_R$ to have a single parameter p bounding the value variable val. Then, $\mathcal{A}_R[p]$ accepts $v \oplus w$ iff there is a run of $\mathcal{A}_R$ whose value varible does not exceed p and $\inf\{p : \mathcal{A}_R[p] \text{ accepts } v \oplus w\}$ is the value of $v \oplus w$ assigned by $\mathcal{A}_R$.

Assume that $\mathcal{A}_R$ *contains the timed identity*, i.e., for every timed word v, $\mathcal{A}_R[\vec{0}]$ accepts $v \oplus v$. Observe that d_M defined by $d_M(w) = \inf\{|\vec{p}| : v \in \mathcal{L}(\mathcal{A}_M), \mathcal{A}_R[\vec{p}] \text{ accepts } v \oplus w\}$ is an automatic similarity measure. Clearly, for every M', $d_M(M') \geq 0$ and since $\mathcal{A}_R$ contains the timed identity, $d_M(M) = 0$. The class of timed languages accepted by hybrid automata is closed under projection, hence there is a hybrid automaton computing d_M. Furthermore, it can be defined by a hypervisor in a uniform way:

Proposition 26 *Let R be a hybrid-automatic weighted relation computed by $\mathcal{A}_R$ and let $\mathcal{A}_M$ be a hybrid automaton. The similarity measure defined from $\mathcal{A}_M$ by R can be defined by a hypervisor that has a symbolic definition that uses components of $\mathcal{A}_M$ and $\mathcal{A}_R$ as predicates. Moreover, if $\mathcal{A}_R, \mathcal{A}_M$ are weighted timed automata (compact initialized rectangular automata), the automaton $\mathcal{A}_M \ltimes H_R$ is a weighted timed automaton (compact initialized rectangular automaton).*

Example 27 *Consider the model from Example 23 and the question of the maximal latency of the sensor tolerated by the model. We decouple a heartbeat event into two consecutive events: an actual heartbeat and an observed heartbeat. Then, we specify that the trace of actual heartbeats and observed heartbeats are in a weighted relation R_d defined as follows. The weighted relation R_d is defined only on pairs of* 1-interleaved *timed-words, i.e., timed words w, v satisfying that between any two events of w, there is at most one event of v and vice versa. For timed words w, v that are* 1*-interleaved, $untime(w) = untime(v)$ and every event of w is earlier than its counterpart in v, the value of wRv is the maximal delay between any event of w and its counterpart in v. Otherwise, $wRv = \infty$. We show in Section 6, that R_d is a timed-automatic weighted relation.*

6 Hybrid-automatic weighted relations

We have seen in Section 5.2 that hybrid-automatic weighted relations conveniently express model-independent similarity measures. This motivates the study of their expression power. We show basic construction methods as well as limitations of hybrid-automatic weighted relations. However, the fact that certain weighted relations are not expressible by hybrid automata does not imply that the similarity measures defined by those relations are not hybrid-automatic.

Hybrid-automatic (weighted) relations are substantially different than automatic relations on words (defined in Section 6.2). A key difference between words and timed words is that events in the latter can be arbitrarily dense, i.e., there can be arbitrarily many events in a fixed time interval. In consequence, in the disjoint union $w \oplus v$, there is no bound on the number of events from v between two consecutive events of w and vice versa. As we will see in Theorem 37 this leads to undecidability of simple (in the word case) problems. To avoid that, we define the notion of K-interleaved words, which intuitively means that words are synchronized.

6.1 Compositionality of hybrid-automatic relations

Composition and intersection are two simple constructions that build new hybrid-automatic weighted relations from the ones already defined.

Definition 28 *Let R, S be weighted relations. We define the* composition $R \circ S$ *and* intersection $R \cap S$*, as follows. For all timed words w_1, w_2:*

$$w_1(R \circ S)w_2 = \inf\{w_1 R w_3 + w_3 S w_2 : w_3 \text{ is a timed word}\}$$
$$w_1(R \cap S)w_2 = w_1 R w_2 + w_1 S w_2$$

Proposition 29 *The composition and intersection of hybrid-automatic weighted relations are hybrid-automatic weighted relations.*

Moreover, if the relations R, S are computed by weighted timed automata (compact initialized rectangular automata), then $R \circ S$ and $R \cap S$ are computed by weighted timed automata (compact initialized rectangular automata).

6.2 K-interleaved timed words

In this section, we define K-interleaved property, which intuitively expresses that two words are synchronized. Next, we show that discrete automatic relations can be lifted to the hybrid case assuming that the timed words are K-interleaved. That assumption is essential (Theorem 37).

Definition 30 *Let w_1, w_2 be timed words over disjoint alphabets. We say that w_1, w_2 are K-interleaved iff in any time interval $[t_1, t_2]$ the numbers of events from w_1 and from w_2 differ by at most K.*

We briefly introduce automatic relations on discrete words. The *convolution* of words w_1, w_2, denoted by $w_1 \otimes w_2$, is a word over $(\Sigma \cup \{\#\}) \times (\Sigma \cup \{\#\})$ of length $max(|w_1|, |w_2|)$ such that the i-th letter of $w_1 \otimes w_2$ is a pair of the i-th letters of w_1 and w_2. If $|w_j| < i$, we assume that its i-th letter is $\#$. E.g. $ab \otimes c = (a, c)(b, \#)$. A relation R on $\Sigma^* \times \Sigma^*$ is *automatic* iff there is a finite word automaton $\mathcal{A}_R$ such that for all v, w, vRw holds iff $\mathcal{A}$ accepts $v \otimes w$.

Proposition 31 *Let $K > 0$ and let S be a discrete automatic relation. A weighted (hybrid) relation R defined for all w_1, w_2 as $w_1 R w_2 = 0$, if w_1, w_2 are K-interleaved and $untime(w_1)$ S $untime(w_2)$, and $w_1 R w_2 = \infty$ otherwise, is a hybrid-automatic weighted relation and it is computed by a monotonic timed automaton.*

A weighted relation R on $\Sigma^* \times \Sigma^*$ is *automatic* iff there is a weighted finite word automaton $\mathcal{A}_R$ such that for all v, w, vRw equals to the value of $v \otimes w$ assigned by $\mathcal{A}$.

Proposition 32 *Let $K > 0$ and let S be an automatic weighted relation. Define a weighted relation R on timed words as follows: for all w_1, w_2, if w_1, w_2 are K-interleaved, $w_1 R w_2 = untime(w_1) S untime(w_2)$, otherwise $w_1 R w_2 = \infty$. The relation R is a hybrid-automatic weighted relation and it is computed by a monotonic weighted timed automaton.*

Propositions 31 and 32 have virtually the same proofs. The timed automaton computing R neglects time and simulates the automaton computing S. Corresponding events of two timed words may appear at different times, but since w_1, w_2 are K-interleaved, the timed automaton has to remember only last K events to synchronize w_1 and w_2.

Definition 33 *Let $K > 0$ and let z be a continuous variable. We define distances $d_K^{\infty,z}, d_K^{1,z}$ on timed words $w_1 = \langle a_1^1, \nu_1^1\rangle\langle a_2^1, \nu_2^1\rangle \ldots \langle a_k^1, \nu_k^1\rangle$, $w_2 = \langle a_1^2, \nu_1^2\rangle\langle a_2^2, \nu_2^2\rangle \ldots \langle a_l^2, \nu_l^2\rangle$ as follows: if $k = l$ and w_1, w_2 are K-interleaved:*

$$w_1 dt_K^{\infty,z} w_2 = \max_{i \in \{1,\ldots,k\}} |\nu_i^1(z) - \nu_i^2(z)|$$
$$w_1 dt_K^{1,z} w_2 = \sum_{i \in \{1,\ldots,k\}} |\nu_i^1(z) - \nu_i^2(z)|$$

Otherwise, $w_1 dt_K^{\infty,z} w_2 = w_1 dt_K^{1,z} w_2 = \infty$.

Proposition 34 *For every $K > 0$ and every variable z, the relations $dt_K^{\infty,z}, dt_K^{1,z}$ are hybrid-automatic weighted relations. If z is the time variable, then $dt_K^{\infty,z}, dt_K^{1,z}$ can be computed by, respectively, a monotonic timed automaton and a monotonic weighted timed automaton.*

Example 35 *Observe that the relation from Example 27 is computed by a monotonic timed automaton. Indeed, that relation is an intersection of three relations $dt_1^{\infty,x}, \mathrm{Id}_1$ and $\geq_1$ defined as follows. $dt_1^{\infty,x}$ is a weighted relation from Definition 33. It is computed by a monotonic timed automaton (Proposition 34). The relation $w\mathrm{Id}_1 v$ is defined by $w\mathrm{Id}_1 v = 0$ iff w, v are 1-interleaved and $untime(w) = untime(v)$, and $w\mathrm{Id}_1 v = \infty$ otherwise. It is computed by a monotonic timed automaton (Proposition 31 applied to the identity relation.) Finally, the relation $\geq_1$ is defined by $w\geq_1 v = 0$ iff w, v are 1-interleaved and every event of w is earlier than its counterpart in v, and $w\geq_1 v = \infty$ otherwise. It is computed by a monotonic timed automaton that counts the difference of events from w and from v. Clearly, the difference of events from w and from v is always 0 or 1 iff w, v are 1-interleaved and every event of w is earlier than its counterpart in v.*

Example 36 *(Skorohod distance) The Skorohod distance between functions $x, y : \mathbb{R}^n \mapsto \mathbb{R}$ is defined as $d_S(x, y) = \inf\{\|\mathrm{Id} - \Lambda\|_\infty + \|x - y \circ \Lambda\|_\infty : \Lambda$ is a bijective continuous function$\}$. It has been pointed out in [6], that d_S is the right measure to compare hybrid systems. The intuition behind the Skorohod distance is that Λ represents distorted time flow, and the distance measures the optimal balance between the distortion of time ($\|\mathrm{Id} - \Lambda\|_\infty$) and space ($\|x - y \circ \Lambda\|_\infty$).*

The Skorohod distance can be expressed by an affine hybrid automaton on K-interleaved timed words. Observe that for all w, v, $d_S(w, v) = w\, dt_K^{\infty,t} \circ (dt_K^{\infty,z} \cap \mathrm{tId})\, v$, where tId *is defined as $w\,\mathrm{tId}\, v = 0$ if projections of w, v on the time variable are equal, and $w\,\mathrm{tId}\, v = \infty$ otherwise. Clearly,* tId *is recognized by a timed automaton.*

Unfortunately, a hybrid-automaton expressing $dt_K^{\infty,z}$ is neither rectangular nor initialized.

6.3 Inexpressible hybrid relations

Propositions 31, 32 and 34 refer only to timed words that are K-interleaved. In a way, it is clear that automata with finite memory are not able to process meaningfully piling up events from one of the timed words. However, the question arises, is there a (more general) notion of hybrid automata whose emptiness problem is decidable, but it is strong enough to define nontrivial hybrid-automatic weighted relations without assuming that timed words are K-interleaved? The following theorem proves otherwise.

Theorem 37 *Let* Id *be a weighted relation defined as follows: for all timed words w_1, w_2, $w_1 \mathrm{Id} w_2 = 0$ iff $untime(w_1) = untime(w_2)$, and $w_1\,\mathrm{Id}\, w_2 = \infty$ otherwise.*

The problem, given a hybrid-automatic weighted relation R, decide whether there are w_1, w_2 such that $w_1(\mathrm{Id} \cap R)w_2$ equals 0, is undecidable.

7 Conclusions

In this paper we presented the model-measuring framework for the hybrid case, where distances are represented by parametrized hybrid automata. The theory developed in this paper applies to any class of hybrid automata, but the model-measuring problem is decidable only on special classes of hybrid automata. We give two examples of such classes, weighted timed automata and compact initialized rectangular automata.

Our future work is to extend the class of hybrid automata for which the model-measuring problem is decidable.

Acknowledgment

This work was supported in part by the Austrian Science Fund NFN RiSE (Rigorous Systems Engineering) and by the ERC Advanced Grant QUAREM (Quantitative Reactive Modeling).

References

[1] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A Henzinger, P-H Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical computer science*, 138(1):3–34, 1995.

[2] Rajeev Alur and P. Madhusudan. Decision problems for timed automata: A survey. In *SFM*, volume 3185 of *LNCS*, pages 1–24. Springer, 2004.

[3] Tawhid Bin Waez, Juergen Dingel, and Karen Rudie. A survey of timed automata for the development of real-time systems. *Computer Science Review*, 2013.

[4] Patricia Bouyer, Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On the optimal reachability problem on weighted timed automata. *FMSD*, 31(2):135–175, October 2007.

[5] Thomas Brihaye, Laurent Doyen, Gilles Geeraerts, Joël Ouaknine, Jean-François Raskin, and James Worrell. On reachability for hybrid automata over bounded time. In *Automata, Languages and Programming*, pages 416–427. Springer, 2011.

[6] Paul Caspi and Albert Benveniste. Toward an approximation theory for computerised control. In *Embedded Software*, pages 294–304. Springer, 2002.

[7] Goran Frehse, Sumit Kumar Jha, and Bruce H Krogh. A counterexample-guided approach to parameter synthesis for linear hybrid automata. In *HSCC*, pages 187–200. Springer, 2008.

[8] Laurent Fribourg and Ulrich Kühne. Parametric verification and test coverage for hybrid automata using the inverse method. *Int. J. Found. Comput. Sci.*, 24(2):233–250, 2013.

[9] Thomas A Henzinger. *The theory of hybrid automata*. Springer, 2000.

[10] Thomas A Henzinger, Peter W Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? In *STOC*, pages 373–382. ACM, 1995.

[11] Thomas A. Henzinger and Jan Otop. From model checking to model measuring. In *CONCUR*, volume 8052 of *LNCS*, pages 273–287. Springer, 2013.

[12] Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits Vaandrager. Linear parametric model checking of timed automata. *The Journal of Logic and Algebraic Programming*, 52:183–220, 2002.

[13] Joseph S. Miller. Decidability and complexity results for timed automata and semi-linear hybrid automata. In *HSCC*, volume 1790 of *LNCS*, pages 296–309. Springer, 2000.

8 Proofs from Section 3

Proposition 7 *Monotonicity of parametric timed automata is undecidable.*

Proof: The problem, given a parametric timed automaton $\mathcal{A}[q]$ decide whether there is a $q \in (1,2)$ such that $[\mathcal{A}[q]]$ is non-empty, is undecidable [13]. We will reduce that problem to monotonicity.

Let $\mathcal{A}[q]$ be a parametric timed automaton. One can easily construct a parametric timed automaton $\mathcal{B}[q]$ such that $[\mathcal{B}[q]]$ is nonempty iff $q \in (1,2)$. Observe that $(\mathcal{A} \times \mathcal{B})[q]$ is monotonic iff for every $q \in (1,2)$, $[\mathcal{A}[q]]$ is empty. Indeed, $[\mathcal{A}]$ is empty for every $q \in (1,2)$, then $[(\mathcal{A} \times \mathcal{B})[q]]$ is empty for every q. Thus, it is monotonic. Conversely, if $[\mathcal{A}[q]]$ is nonempty for some $q \in (1,2)$, $[(\mathcal{A} \times \mathcal{B})[q]]$ is nonempty as well. But, $[(\mathcal{A} \times \mathcal{B})[2]]$ is empty, thus $\mathcal{A} \times \mathcal{B}$ is not monotonic. □

Proposition 9 *Every weakly decidable and f-bounded class of monotonic hybrid automata $\mathfrak{C}$ admits parameter approximation in time $O((f(|\mathcal{A}|)+n) \cdot T(|\mathcal{A}| + PM(\mathcal{A}) \cdot (log(f(|\mathcal{A}|))+n)))$, where $T(\cdot)$ is the complexity of the emptiness problem for $\mathfrak{C}$.*

Proof: First, let $\vec{p}_0^{\top} = (f(|\mathcal{A}|), \ldots, f(|\mathcal{A}|))$. Due to f-boundedness and monotonicity, if there is $\vec{q}$ such that $[\mathcal{A}[\vec{q}]] \neq \emptyset$, then $[\mathcal{A}[\vec{p}_0^{\top}]] \neq \emptyset$. Thus, if $[\mathcal{A}[\vec{p}_0^{\top}]] = \emptyset$ we can immediately return $\emptyset$. Otherwise, $\vec{p}_0^{\perp} = \vec{0}$ and by binary search, we define a sequence of pairs $\langle \vec{p}_0^{\perp}, \vec{p}_0^{\top} \rangle, \langle \vec{p}_1^{\perp}, \vec{p}_1^{\top} \rangle$ so that $[\mathcal{A}[\vec{p}_i^{\top}]] \neq \emptyset$ $[\mathcal{A}[\vec{p}_i^{\perp}]] = \emptyset$, until for some k, $[\mathcal{A}[\vec{p}_0^{\perp}]] \neq \emptyset$ or $|\vec{p}_k^{\top} - \vec{p}_k^{\perp}| < 2^{-n}$. Such k is bounded by $log(f(|\mathcal{A}|)) + k$ multiplied by the length of $\vec{p}$. All bounds from the statement can be easily verified. □

Proposition 10 *Let $\mathfrak{C}$ be a weakly decidable class of monotonic hybrid automata. The class $\mathfrak{C}$ is strongly decidable iff there is a computable function f such that $\mathfrak{C}$ is f-bounded.*

Proof: Proposition 9 give the implication from right to left. The converse implication: Consider a function $g : \mathfrak{C} \mapsto \mathbb{R}$, such that for a given parametric automaton $\mathcal{A}$, it returns $|\vec{p}|$ such that $\mathcal{A}[\vec{p}] \neq \emptyset$, if there are such parameters and 0 otherwise. The function g is effectively computable. It first checks whether there are parameters $\vec{p}$ such that $\mathcal{A}[\vec{p}] \neq \emptyset$. If there are, it check vectors of successive natural numbers, $\vec{0}, \vec{1}, \ldots$. Due to monotonicity, such a procedure terminates. Otherwise, it returns 0. Since $\mathfrak{C}$ is strongly decidable, the first check is decidable. In consequence, g is computable.

Observe that $f(n)$ defined as $max\{g(\mathcal{A}) : \mathcal{A} \in \mathfrak{C}, |\mathcal{A}| \leq n\}$ is the function that gives a bound on parameters and it is computable. □

Proposition 11 *The class of monotonic weighted timed automata (over finite words) is exponentially bounded.*

Proof: Let $\mathcal{A}$ be a monotonic timed automaton with multiple parameters $\vec{p}$. Let us fix a total order on parameters, e.g. $p_0 \preceq p_1 \preceq \ldots \preceq p_k$. We say that p_i has a higher order than p_j iff $p_j \prec p_i$.

We call a constraint *positive* (resp. *negative*) iff it is of one the forms $x - s(\vec{p}) \leq 0$, $x - s(\vec{p}) < 0$ $x + s(\vec{p}) \geq 0$ or $x + s(\vec{p}) > 0$ and the parameter in s with the highest order that occurs with a non-zero coefficient, occurs with positive (resp. negative) coefficient. Each parametric constraint is either positive or negative.

Let $\mathcal{A}^+$ be obtained from $\mathcal{A}$ by removing all transitions with negative constraints. Observe that for each finite timed word accepted by $\mathcal{A}$ with some parameters has a run (with some parameters) that avoids all transitions with negative constraints. Indeed, for sufficiently large parameters with sufficiently large difference between them, we can make all negative constraints unsatisfiable, i.e., in constraints of the forms $x - s(\vec{p}) \leq 0$ or $x - s(\vec{p}) < 0$, $s(\vec{p})$ will be less than 0 and in constraints of the forms $x + s(\vec{p}) \geq 0$ or $x + s(\vec{p}) > 0$, $-s(\vec{p})$ will be greater than the last timestamp of the timed word w. Thus, the unions over all parameters $\vec{p}$ of $[\mathcal{A}[p]]$ and $[\mathcal{A}^+[p]]$ are equal. Therefore, if there is $\vec{p}_1$ such that $[\mathcal{A}[\vec{p}_1]] \neq \emptyset$, there is also $\vec{p}_2$ such that $[\mathcal{A}^+[\vec{p}_2]] \neq \emptyset$. Therefore, we assume that such $\vec{p}_2$ exists.

Consider $\mathcal{A}^{\top}$ resulting from $\mathcal{A}^+$ by replacing each constraint containing a parameter with $\top$, the propositional constant true. As $\mathcal{A}^{\top}$ has no parameters, it is a usual timed automaton. Observe that if $[\mathcal{A}^{\top}]$ is nonempty, then it has an accepting run of exponential duration, i.e., an accepting run $\pi = \langle l_0, t_0 \rangle \ldots \langle l_n, t_n \rangle$, where $t_0 = 0$ and t_n is exponential w.r.t. $|\mathcal{A}^{\top}|$. Indeed, an accepting run of minimal length in the region graph of $\mathcal{A}^{\top}$ visits each region at most once and it stays in each region at most one unit of time. There are exponentially many regions in the region graph, therefore the duration of a shortest accepting run is exponentially bounded. We show that with appropriate parameters π is also an accepting run in $\mathcal{A}^+$ and $\mathcal{A}$.

Let c be the maximal absolute value of all coefficients occurring in constraints of $\mathcal{A}^+$. Consider the following instantiation $\vec{p}$ of parameters: $p_0 = c \cdot (t_n + \epsilon), p_1 = 2 \cdot c \cdot (t_n + \epsilon), \ldots, p_k = 2^k \cdot c \cdot (t_n + \epsilon)$. There are two types of constraints in $\mathcal{A}^+$ and they are both satisfied on π:

1. the constraints of the form $x \leq s(\vec{p})$ or $x - s(\vec{p}) < 0$, where the parameter in t with the highest order occurs with positive coefficient are satisfied on π. Observe that $t_n + \epsilon \leq s(\vec{p})$. The constraints $x \leq t_n + \epsilon$ and $x < t_n + \epsilon$ are satisfied on π.

2. the constraints of the form $x \geq s(\vec{p})$ or $x > s(\vec{p})$, where the parameter in t with the highest order occurs with negative coefficient are satisfied on π. That is because $-(t_n + \epsilon) \geq s(\vec{p})$, therefore they are of the forms $x \geq -\epsilon$ and $x > -\epsilon$. As clocks are always non-negative, those constraints are always satisfied.

Therefore, π is an accepting run of $\mathcal{A}^+[\vec{p}]$. Since every run of $\mathcal{A}^+[\vec{p}]$ is a run of $\mathcal{A}$, π is an accepting run of $\mathcal{A}[\vec{p}]$ and all components of $\vec{p}$ are exponentially bounded in $|\mathcal{A}|$.

For weighted timed automata, observe that the weight is linearly bounded in the duration of the run and the number of events. □

Proposition 12 *The class of monotonic compact initialized rectangular automata (over finite words) is exponentially bounded.*

Proof: Let $\mathcal{A}$ be a compact initialized rectangular automaton without parameters. Due to [10], there is a timed automaton $\mathcal{B}$ whose sets of states and variables are linear in $|\mathcal{A}|$ and the constants that appear in its constraints appear in $\mathcal{A}$ as well. Thus, such a timed automaton $\mathcal{B}$ has exponential region graph. Hence, if $[\mathcal{A}] = [\mathcal{B}]$ is nonempty, there is a timed word of exponential duration, i.e., a timed word $(a_0, t_0), \ldots, (a_k, t_k)$ with $t_k - t_0$ exponentially bounded in $|\mathcal{A}|$.

Assume that $[\mathcal{A}]$ is nonempty and $w \in [\mathcal{A}]$ is a timed word of exponential duration. Since $\mathcal{A}$ is compact, there is a bound C on all its flows. Then, during the run of $\mathcal{A}$ on w, the values of all variables are contained in $[-C \cdot d_w, C \cdot d_w]$, where d_w is the duration of the timed word w.

Next, we can repeat the proof of Proposition 11. □

9 Proofs from Section 5

Proposition 26 *Let R be a hybrid-automatic weighted relation computed by $\mathcal{A}_R$ and let $\mathcal{A}_M$ be a hybrid automaton. The similarity measure defined from $\mathcal{A}_M$ by R can be defined by a hypervisor that has a symbolic definition that uses components of $\mathcal{A}_M$ and $\mathcal{A}_R$ as predicates. Moreover, if $\mathcal{A}_R, \mathcal{A}_M$ are weighted timed automata (compact initialized rectangular automata), the automaton $\mathcal{A}_M \ltimes H_R$ is a weighted timed automaton (compact initialized rectangular automaton).*

Proof: To see that, consider a monotonic hybrid automaton $\mathcal{A}_R$ over $\Sigma \times \{1, 2\}$ that represents a hybrid-automatic weighted relation R. A hypervisor $H_R = (\mathcal{A}_H, \tau_H)$ is defined as follows: $\mathcal{A}_H$ in over the alphabet Σ; on w it simulates execution of $\mathcal{A}_R$ on $v \oplus w$ by guessing v step by step. It remembers the currently guessed letter in its state and whenever it takes an ϵ-transition according to a guessed letter a, the hypervised automaton $\mathcal{A}_M$ takes an ϵ-transition, defined by τ_H, consistent with the transition that it would take if it read a. In other words, H defines blind a transitions as defined in Section 5. Such transitions can be symbolically defined.

A quick analysis of the semi-direct product shows that this construction preserves classes of weighted timed automata and compact initialized hybrid automata. □

10 Proofs from Section 6

Proposition 29 *The composition and intersection of hybrid-automatic weighted relations are hybrid-automatic weighted relations.*

Proof: Let $\mathcal{A}_R$, $\mathcal{A}_S$ be monotonic hybrid automata computing R and S.

Intersection: We can assume that parameters of $\mathcal{A}_R$ and $\mathcal{A}_S$, $\vec{p}_R, \vec{p}_S$, are disjoint. Observe that the automaton $\mathcal{A}_R \times \mathcal{A}_S$ with the vector norm $|\vec{p}| = |\vec{p}_R|_R + |\vec{p}|_S$ computes $R \cap S$.

Composition: We can assume that parameters of $\mathcal{A}_R$ and $\mathcal{A}_S$, $\vec{p}_R, \vec{p}_S$, are disjoint. The automata $\mathcal{A}_R$, $\mathcal{A}_S$ work on $w \oplus v$ and they distinguish events from w and v by a second component, i.e., an event (a, ν) from w becomes $(\langle a, i\rangle, \nu_i)$,, where $\nu_i : X \times \{i\} \cup \{t\} \mapsto \mathbb{R}$ is defined as $\nu_i(\langle x, i\rangle) = \nu(x)$ and $\nu_i(t) = \nu(t)$.

We can relabel letters and variables in $\mathcal{A}_R$, $\mathcal{A}_S$ to $\mathcal{A}'_R$, $\mathcal{A}'_S$ such that:

(i) letters $\Sigma \times \{1\}$ and variables $X \times \{1\}$ in $\mathcal{A}_R$ remain the same in $\mathcal{A}'_R$

(ii) letters $\Sigma \times \{2\}$ and variables $X \times \{2\}$ in $\mathcal{A}_R$ are relabeled to $\Sigma \times \{3\}$ and variables $X \times \{3\}$ in $\mathcal{A}'_R$

(iii) letters $\Sigma \times \{1\}$ and variables $X \times \{1\}$ in $\mathcal{A}_S$ are relabeled to $\Sigma \times \{3\}$ and variables $X \times \{3\}$ in $\mathcal{A}'_S$

(iv) letters $\Sigma \times \{2\}$ and variables $X \times \{2\}$ in $\mathcal{A}_S$ remain the same in $\mathcal{A}'_S$

Finally, we defined the automaton $\mathcal{A}_{R \circ S}$ as the result of projecting out $\Sigma \times \{3\}$ and variables $X \times \{3\}$.

It is easy to see that the automaton $\mathcal{A}_{R \circ S}$ with the vector norm $|\vec{p}| = |\vec{p}_R|_R + |\vec{p}|_S$ computes $R \circ S$.
□

Proposition 34 *For every $K > 0$ and every variable z, the relations $dt_K^{\infty,z}, dt_K^{1,z}$ are hybrid-automatic weighted relations. If z is the time variable, then $dt_K^{\infty,z}, dt_K^{1,z}$ can be computed by, respectively, a monotonic timed automaton and a monotonic weighted timed automaton.*

Proof: A hybrid automaton computing $w_1 dt_K^{\infty,z} w_2$ works as follows. It implements a FIFO queue buffer the values of z from unmatched events. The queue stores values of z from w_1 or w_2. Assume that it stores values from w_i. If the next event is from w_i, it enqueues its value of z. Otherwise, it dequeues a value of z and checks if the dequeued value of z and the value from the current event differ by at most p. If not it rejects. Otherwise, it continues.

Due to K-interleaved assumption, the queue size does not exceed K. Thus, we can implement it using K continuous variables.

A hybrid automaton computing $w_1 dt_K^{1,z} w_2$ works in a similar way, but instead of comparing values of z, it adds the difference to the weight variable.

Observe that hybrid automata computing $w_1 dt_K^{\infty,z} w_2$ and $w_1 dt_K^{1,z} w_2$ are affine, but they are neither initialized nor rectangular.

Now, we consider a special case, where z is the timed variable. A hybrid automaton $\mathcal{A}_1$ computing $w_1 dt_K^{\infty,t} w_2$ works in a similar way as $w_1 dt_K^{\infty,z} w_2$, but the queue stores clocks. When an event enqueues its value, a corresponding clock is reset. We the event is dequeued, the corresponding clock is checked whether it does not exceed p. Clearly, $\mathcal{A}_1$ is a timed automaton as all variables have slope 1, they are only reset to 0 and the only constraints are of the form $x < p$.

Finally, we construct a weighted timed automaton $\mathcal{A}_2$ that computes $w_1 dt_K^{1,t} w_2$. That automaton has $K + 2$ states $q_0, \ldots, q_K$ and q_{fail}, each of $q_0, \ldots, q_K$ corresponds to the number of unmatched events, and q_{fail} denotes that the automaton detected that w_1, w_2 are not K-interleaved. It does not have weights on transitions, but it has weights associated with the states. For $i \in \{0, \ldots, K\}$, the state q_i has the weight i, i.e., staying in the state q_i for time Δ costs $\Delta \cdot i$. Thus, the automaton needs to remember the number of unmatched events, i.e., the difference between the numbers of events of w_1 and w_2 read so far, i.e., if the automaton has read 3 events from w_1 and none of w_2, then it is in the state q_3.

It can be shown by induction on N that the automaton $\mathcal{A}$ correctly computes $dt(w_1, w_2)$ for all K-interleaved timed words w_1, w_2 with $|w_1| = |w_2| = N$.

□

11 The proof of Theorem 37

Definition 38 *Let $\Delta > 0$ and let R be a hybrid-automatic weighted relation. We say that R piecewise-defines S on Δ-intervals if*

$$w_1 S w_2 = \sum_{j=0}^{\infty} (w_1 \cap [j \cdot \Delta, (j+1) \cdot \Delta]) R (w_2 \cap [j \cdot \Delta, (j+1) \cdot \Delta])$$

We leave the following proposition without a proof.

Proposition 39 *For every rational $\Delta > 0$, every relation piecewise-defined on Δ-intervals by a hybrid-automatic weighted relation is a hybrid-automatic weighted relation.*

Theorem 37 *Let* Id *be a weighted relation defined as follows: for all timed words* w_1, w_2, $w_1 \mathrm{Id} w_2 = 0$ *iff* $untime(w_1) = untime(w_2)$, *and* $w_1 \,\mathrm{Id}\, w_2 = \infty$ *otherwise.*

The problem, given a hybrid-automatic weighted relation R, *decide whether there are* w_1, w_2 *such that* $w_1(\mathrm{Id} \cap R)w_2$ *equals* 0, *is undecidable.*

Proof: Consider an instance of PCP $\Gamma = \{(v_1, w_1), \ldots, (v_k, w_k)\}$. Let R_Γ be a relation defined as follows: for all u_1, u_2, $R_\Gamma(u_1, u_2) = 0$ if for every $j \in \mathbb{N}$ $(untime(u_1 \cap [j, j+1]), untime(u_2 \cap [j, j+1])) \in \Gamma$ and $u_1 R u_2 = \infty$ otherwise. Fact 39 implies that R_Γ is hybrid-automatic weighted relation.

Assume that $u_1(\mathrm{Id} \cap R_\Gamma)u_2 = 0$. Then, $untime(u_1) = untime(u_2)$ and there is a sequence $i_1, \ldots, i_m$ such that $untime(u_1) = v_{i_1} \ldots v_{i_m}$, $untime(u_2) = w_{i_1} \ldots w_{i_m}$. Thus, the instance Γ of PCP has a solution.

Conversely, assume that Γ has a solution $i_1, \ldots, i_m$. Let u_1, u_2 be timed words such that for every $j \in \{1, \ldots, m\}$, $untime(u_1 \cap [j, j+1)) = v_j$ and $untime(u_2 \cap [j, j+1)) = w_j$. Clearly, $u_1 R_\Gamma u_2 = 0$ and $untime(u_1) = untime(u_2)$, hence $u_1 \mathrm{Id} u_2 = 0$. It follows that $u_1(\mathrm{Id} \cap R)u_2 = 0$.

Thus, $u_1(I \cap R_\Gamma)u_2 = 0$ if and only if Γ has a solution. □