A nonarchimedian discretization for timed languages

Cătălin Dima

ENSEIRB, 1, avenue du dr. Albert Schweitzer, Domaine Universitaire - BP 99 33402 Talence Cedex, France

Abstract. We give a new discretization of behaviors of timed automata. In this discretization, timed languages are represented as sets of words containing action symbols, a clock tick symbol 1, and two *delay* symbols δ^- (*negative* delay) and δ^+ (*positive* delay). Unlike the region construction, our discretization commutes with intersection. We show that discretizations of timed automata are, in general, context-sensitive languages over $\Sigma \cup \{1, \delta^+, \delta^-\}$, and give a class of automata that equals the class of languages that are discretizations of timed automata, and show that their emptiness problem is decidable.

1 Introduction

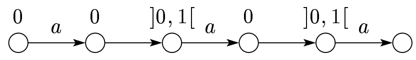
Timed automata [AD94] are a successful and widely used extension of finite automata for modeling real-time systems. They are finite automata endowed with clocks that measure time passage. Clocks are real-valued variables and evolve synchronously at rate 1. Transitions are guarded by arithmetic conditions on the clocks, and some transitions might reset some clocks to 0.

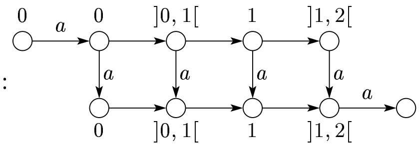
The essential construction for timed automata is the *region construction*, a finite abstraction of the infinite state-space of the automaton. This abstraction preserves emptiness and is the basis of both theoretic results concerning monoid recognizability, and of tools for model-checking timed systems [Yov98,LPY97]. Recently, researchers have shown interest in laying the foundations of a formal timed language theory. Some important progress has been reported on adapting regular expressions to timed automata [BP99,BP02,ACM02,AD03], on adapting monoid recognizability to timed languages [BPT01], and on pumping lemmas for timed languages [Her98,Bea98].

The research reported here has started from the observation that the region construction is not compositional w.r.t. intersection. That is, the region automaton of an intersection of languages of two timed automata is not the intersection of the region automata of the two timed automata. E.g., take the timed languages

$$L_1 = \{t_1 a t_2 \mid t_1, t_2 \in]0,1[\} = \left\| \langle \underline{\mathbf{t}} \rangle_{]0,1[} \langle a \underline{\mathbf{t}} \rangle_{]0,1[} \right\|, \qquad L_2 = \{t_1 a t_2 \mid t_1 + t_2 \in]1,2[\} = \left\| \langle \underline{\mathbf{t}} a \underline{\mathbf{t}} \rangle_{]1,2[} \right\|$$

Here we have used timed regular expressions similar to [ACM02], in which $\underline{\mathbf{t}}$ represents time passage and the *time binding* construction $\langle E \rangle_I$ requires that each behavior generated by expression E be observed within the interval I. Here are region constructions for these languages:

$\mathcal{A}_1$: 

$\mathcal{A}_2$: 

Our observation is that the classical "intersection construction" applied to the finite automata $\mathcal{A}_1$ and $\mathcal{A}_2$ does not yield a region automaton for the timed language $L_1 \cap L_2$. Rather, we need to construct first a timed automaton for $L_1 \cap L_2$ and then to discretize it into a region automaton.

We present here a different discretization for timed automata, as (sets of) words over an alphabet containing action symbols, “clock tick” symbols and two special symbols δ^- and δ^+ denoting positive, resp. negative delay. The delay symbols give some information about how to “skew” a sequence of clock ticks with some positive or negative delay, in order to obtain dense-time behaviors from a discrete-time information. This way, the set of discretizations is organized as a monoid of words. We believe that this discretization would pave the way for some further imports from formal language theory, due to the richer structure it provides. On the other hand, we show here that this discretization produces context-sensitive languages, fact which may give another explanation on the undecidability of the universality problem for timed automata. The delay symbols we introduce have a “nonarchimedian” semantics, in the sense that the semantics of any number of positive delays is smaller than clock tick. This comes from the following reasons: clock ticks are used for quantitative timing information, while delay symbols are a qualitative timing information, since they “alter” the information provided by clock ticks.

We investigate in this paper a class of automata that is suitable for representing discretizations of languages of timed automata. Our class of automata, called here *oscillator automata*, have the ability to add positive and/or negative delays, and to say whether the result is a positive or a negative delay. However they are not able to count delays. We show here that this class of automata has a decidable emptiness problem, and that the class of languages that are discretizations of timed automata equals the class of languages accepted by oscillator automata.

The paper is divided as follows: in the next section we remind the notions of timed automata and timed regular expressions. In the third section we give the semantics of our discretization, and several basic properties they have. In the fourth section we introduce oscillator automata and show the decidability of their emptiness problem. The fifth section gives the results concerning the connection between oscillator automata and timed automata.

2 Preliminaries

Behaviors of timed systems can be modeled by **timed words** over a set of symbols Σ. Two presentations can be given to (most of the) timed words, one being as sequences of symbols from Σ with time stamps, the other as mixed sequences of symbols from Σ and real numbers. For example, $(a, 3)(b, 3.1)$ and $1\,2\,a\,0.1\,b$ represent the same timed word. A special case occurs when time elapses after the last action – it is the case of the timed word $3\,a\,1\,b\,1$. We will then associate to this the time stamp sequence $(a, 3)(b, 4)(\varepsilon, 5)$, hence accepting that the last tuple in a time stamp sequence be labeled with the empty word. The *untiming* of a timed word is the sequence of actions in it. *Renamings* can be defined on timed words in the straightforward way: for example, $[a \mapsto b](a, 3)(b, 3.1) = (b, 3)(b, 3.1)$.

A **timed automaton** [AD94] is a tuple $\mathcal{A} = (Q, \mathcal{X}, \Sigma, \delta, Q_0, Q_f)$ where Q is a finite set of *states*, $\mathcal{X}$ is a finite set of *clocks*, Σ is a finite set of *action symbols*, $Q_0, Q_f \subseteq Q$ are sets of *initial*, resp. *final* states, and δ is a finite set of tuples (*transitions*) (q, C, a, X, r) where $q, r \in Q$, $X \subseteq \mathcal{X}$, $a \in \Sigma \cup \{\varepsilon\}$ and C is a finite conjunction of *clock constraints* of the form $x \in I$, where $x \in \mathcal{X}$ and $I \subseteq [0, \infty[$ is an interval with integer (or infinite) bounds.

For each transition $(q, C, a, X, r) \in \delta$, the component C is called the *guard* of the transition, a is called the *action label* of the transition, and X is called the *reset component* of the transition. We consider the set of clocks $\mathcal{X} = \{x_1, \ldots, x_n\}$, and identify each reset component X with the subset of indices of the clocks in X, that is, with $\{i \mid i \in [1 \dots n], x_i \in X\}$. Here $[1 \dots n]$ stands for $\{1, \dots, n\}$, and, in general, we denote $[i \dots j] = \{i, i+1, \dots, j\}$ for any $i, j \in \mathbb{Z}$.

The semantics of a timed automaton is given in terms of a *timed transition system* $\mathcal{T}(\mathcal{A}) = (\mathcal{Q}, \theta, \mathcal{Q}_0, \mathcal{Q}_f)$ where $\mathcal{Q} = Q \times \mathbb{R}^n_{\geq 0}$, $\mathcal{Q}_0 = Q_0 \times \{\mathbf{0}_n\}$, $\mathcal{Q}_f = Q_f \times \mathbb{R}^n_{\geq 0}$ and

$$\begin{aligned}\theta = \big\{(q,v) \xrightarrow{t} (q,v') \mid v'_i = v_i + t,\ \forall i \in [1 \dots n]\big\} \cup \\ \big\{(q,v) \xrightarrow{a} (q',v') \mid \exists (q, C, a, X, q') \in \delta \text{ such that } v \models C \text{ and} \\ \text{for all } i \in [n],\ \text{if } i \in X \text{ then } v'_i = 0 \text{ and if } i \notin X \text{ then } v'_i = v_i\big\}\end{aligned}$$

Informally, the automaton can make t-transitions representing time-passage in a state, in which all clocks advance by t, and discrete transitions, in which state changes. The discrete transitions are enabled when the "current clock valuation" v satisfies the guard C of a certain tuple $(q, C, a, X, r) \in \delta$, and when they are executed, the clocks in the "reset component" X are set to zero.

A *run* in $\mathcal{T}(\mathcal{A})$ is a chain $(q^0, v^0) \xrightarrow{\xi_1} (q^1, v^1) \xrightarrow{\xi_2} \dots \xrightarrow{\xi_k} (q^k, v^k)$ of transitions from θ. An *accepting run* in $\mathcal{T}(\mathcal{A})$ is a run which starts in $\mathcal{Q}_0$ and ends in $\mathcal{Q}_f$. The *accepted language* of $\mathcal{A}$ is the set of timed words which label some accepting run of $\mathcal{T}(\mathcal{A})$. Two timed automata are called *equivalent* iff they have the same language.

The class of **timed regular expressions** is built using the following grammar:

$$E ::= 0 \mid \varepsilon \mid \underline{\mathbf{t}}z \mid E + E \mid E \cdot E \mid E^* \mid \langle E \rangle_I \mid E \wedge E \mid [a \mapsto z]E, \tag{1}$$

where $z \in \Sigma \cup \{\varepsilon\}$ and I is an interval.

The semantics of timed regular expressions is in terms of timed words:

$$\begin{aligned}
\|\underline{\mathbf{t}}z\| &= \{tz \mid t \in \mathbb{R}_{\geq 0}\} & \|E_1 + E_2\| &= \|E_1\| \cup \|E_2\| \\
\|E_1 \cdot E_2\| &= \|E_1\| \cdot \|E_2\| & \|\langle E \rangle\|_I &= \{\sigma \in \|E\| \mid \ell(\sigma) \in I\} \\
\|E^*\| &= \|E\|^* & \|0\| = \emptyset\,, \quad \|\varepsilon\| &= \{\varepsilon\} \\
\|E_1 \wedge E_2\| &= \|E_1\| \cap \|E_2\| & \|[a \mapsto z]E\| &= \{w[a \mapsto z] \mid w \in \|E\|\}
\end{aligned}$$

Theorem 1 ([ACM02]). *The class of timed languages accepted by timed automata equals the class of timed languages accepted by timed regular expressions.*

3 Nonarchimedian delays

The basic idea that we exploit is that timed words represented by timed automata can be grouped into *regions* (the name shows the similarity to the regions of [AD94]). An n-dimensional region is a *nonempty* subset of $\mathbb{R}_{\geq 0}{}^n$ which represents the solution to a constraint of the form $\bigwedge_{0 \leq i < j \leq n} x_j - x_i \in I_{ij}$, where I_{ij} is either a point interval or an open unit interval, and $x_0 = 0$.

Remark 1. [Dim02] The constraint $\bigwedge_{0 \leq i < j \leq n} x_j - x_i \in I_{ij}$ has a nonempty semantics iff for all $i < j < k$, $I_{ij} + I_{jk} \supseteq I_{ik}$. This set of inclusions is called the *triangle property*.

Hence, the set of all timed words of length n can be decomposed into an infinite but countable family of sets of the form

$$L = \big\{(a_1,t_1)\dots(a_n,t_n) \mid \forall 0 \le i < j \le n, t_j - t_i \in I_{ij}, t_0 = 0\big\} \tag{2}$$

We will call each such set as a *region timed language*. The n-dimensional subset of reals $\big\{(t_1,\dots,t_n) \mid \forall 0 \le i < j \le n, t_j - t_i \in I_{ij}, t_0 = 0\big\}$ will be called the *underlying region* of L and denoted $\mathcal{U}(L)$. Our aim is then to code all countable families like L as a set of words over an extended set of symbols.

The set of symbols we use is $\Delta = \Sigma \cup \{1, \delta^+, \delta^-\}$ with $1, \delta^+, \delta^- \notin \Sigma$. δ^+ will be called the *positive delay symbol* while δ^- will be called the *negative delay symbol*. For each $w \in \Delta^*$ and $x \in \Sigma \cup \{1, \delta^+, \delta^-\}$ we denote $|w|_x$ the *number of occurrences* of the symbol x in w. Words over Δ will be called **δ-words** while languages in Δ will be called as **δ-languages**. Our aim is to code *timed languages* with *δ-languages*.

Let us present first the intuition behind our encoding by some examples. Consider first the "point region" timed language $L_0 = \big\{t_1ab \mid t_1 = 1\big\} = \|\langle \underline{\mathbf{t}}ab\rangle_1\|$. Obviously, the symbol 1 can be used with the meaning of *clock tick*, hence we may code this language as $1ab$. Observe that time does not elapse between a and b.

Consider now the region timed language $L_1 = \big\{t_1at_2b \mid t_1 + t_2 = 1, t_1, t_2 > 0\big\} = \|\langle\ \underline{\mathbf{t}}a\underline{\mathbf{t}}b\rangle_1\|$. We would like to reuse the encoding $1ab$ by adding some information regarding the delay between a and b, and the delay before a. This information should say that clock ticks must be "adjusted" with positive or negative delays. This will be the role of the delay symbols: δ^+=*positive delay*, that is, the clock tick is "skewed positively", while δ^-=*negative delay*, that is, the clock tick is "skewed negatively". The following δ-word encodes L_1: $\omega_1 = 1\delta^- a\delta^+ b$. This says that, before the action a, the time elapses with at most one clock tick, while in between a and b there must be a positive delay. A third constraint is present here: the number of positive delays equals the number of negative delays in the whole word. This constraint encodes the fact that the duration of the whole behavior is exactly one clock tick.

As another example, the language

$$L_2 = \{t_1at_2at_3b \mid t_i > 0, t_1 + t_2 + t_3 = 2\} = \Big\|\langle\langle\underline{\mathbf{t}}a\rangle_{]0,1[}\langle\underline{\mathbf{t}}a\rangle_{]0,1[}\langle\underline{\mathbf{t}}b\rangle_{]0,1[}\rangle_2\Big\|$$

can be coded by the δ-word $\omega_2 = 1\delta^-\delta^- a\delta^+ a\delta^+ b$ because we need *two* negative delay symbols before the action a to counterbalance the existence of the necessary two positive delays. Observe that, in the δ-word ω_1 above, the sub-language $\langle\underline{\mathbf{t}}a\rangle_{]0,1[}$ was coded by $1\delta^- a$, while in the δ-word ω_2 it is coded by $1\delta^-\delta^- a$. Therefore, we may have several codings for the same timed language.

This example raises the question of how many positive delay symbols overcome a clock tick. In our model, the answer is: none. The delay symbols have a *nonarchimedian* semantics, that is, the semantics of any number of δ^+ is always less than one clock tick. To summarize, the role of the delay symbols is to modify the clock tick durations by some positive or negative amount, an amount which may never be larger than one time unit.

Definition 1. *Consider a δ-word $\omega = a_0\eta_1a_1\eta_2\dots\eta_ka_k$, with $a_i \in \Sigma$ for all $1 \le i \le k-1$, $a_0, a_k \in \Sigma \cup \{\varepsilon\}$ and $\eta_i \in \{1, \delta^+, \delta^-\}^*$ for each $1 \le i \le k$. Then, for each $1 \le i < j \le k+1$, denote $p_{ij} = |\eta_i\dots\eta_{j-1}|_{\delta^+}$, the number of δ^+'s in the word $\eta_i\dots\eta_{j-1}$, and similarly $n_{ij} = |\eta_i\dots\eta_{j-1}|_{\delta^-}$ and $v_{ij} = |\eta_i\dots\eta_{j-1}|_1$. The **semantics** of ω is the set of timed words $w = a_0t_1a_1t_2\dots t_ka_k$ with the following property: for each $1 \le i < j \le k$,*

– *If* $p_{ij} > n_{ij}$ *then* $t_i + \ldots + t_j \in]v_{ij}, v_{ij}+1[$
– *If* $p_{ij} = n_{ij}$ *then* $t_i + \ldots + t_j = v_{ij}$
– *If* $p_{ij} < n_{ij}$ *then* $t_i + \ldots + t_j \in]v_{ij}-1, v_{ij}[$

We say that ω *is the (**nonarchimedian**) **discretization** of its semantics* $||\omega||$.

With this definition, the *nonarchimedianity* property can be stated as follows:

$$\text{For all } n \in \mathbb{N}, ||(\delta^+)^n|| < 1, ||1(\delta^-)^n|| > 0 \tag{3}$$

where the inequality is the extension of $<$ on sets of reals.

Remark 2. Note that not all δ-words have nonempty semantics: $||\delta^-|| = \emptyset$ because we cannot have timed words with negative durations. However, if such "delay less than zero" information does not occur in a δ-word, then that δ-word has a nonempty semantics, as stated in the following proposition, whose proof is given in the appendix.

Proposition 1. *For each* δ*-word* $\omega = a_0\eta_1 a_1\eta_2 \ldots \eta_k a_k$*, we have* $||\omega|| \neq \emptyset$ *if and only if* ω *satisfies the following property:*

$$(*) \text{ For all } i \in [1 \ldots k], \text{ if } ||\eta_i||_1 = 0 \text{ then } ||\eta_i||_{\delta^-} \leq ||\eta_i||_{\delta^+}$$

Remark 3. Note that semantics of δ-words is not compositional w.r.t. concatenation: $||\delta^+ a|| = ||\delta^+\delta^+ a||$, but

$$||\delta^+ a \cdot \delta^- 1|| \neq ||\delta^+\delta^+ a|| \cdot ||\delta^- 1||$$

This has the implication that no choice of a "representative δ-word" for a class of timed language is "good" w.r.t. concatenation.

The first idea for solving this problem is to work with *all* the δ-words that represent a given timed language. Or, in other words, to work with equivalence classes w.r.t. semantics. We will call a δ-language D as *saturated* iff, whenever $\omega \in D$ and $||\omega|| = ||\omega'||$ for some $\omega, \omega' \in \Delta^*$, then $\omega' \in D$. If D is not saturated, we denote $\mathsf{sat}(D)$ the least saturated δ-language that contains D. Let us see first the structure of the saturated δ-language discretizing a region timed language $L = \{(a_1, t_1) \ldots (a_n, t_n) \mid \forall 0 \leq i < j \leq n, t_j - t_i \in I_{ij}, t_0 = 0\}$, in which $a_n = \varepsilon$:

Denote P the underlying region for L and take P' one of the point regions which is "closest" to P. (P' may be equal to P if P is a point region). The constraint corresponding to P' can be built by choosing, for each $0 \leq i < j \leq n$, I'_{ij} as either $\{\inf I_{ij}\}$ or $\{\sup I_{ij}\}$, and by checking whether the family $(I_{ij})_{i,j}$ satisfies the triangle property (see the remark 1; note that, in fact, we will get that $I'_{ij} + I'_{jk} = I'_{ik}$). $I'_{i-1,i}$ will give the number of clock ticks in between the $(i-1)$-th action symbol and the i-th action symbol, with the convention that the 0 action symbol is the empty word. We will consider its coding in unary, as $\underbrace{11\ldots1}_{\alpha_{i-1,i}}$.

The addition of positive and negative delay symbols is then done as follows: for each $0 \leq i < j \leq n$, we define

$$J_{ij} = \begin{cases} 0 & \text{when } I'_{ij} = I_{ij} \\ ]0, \infty[& \text{when } I'_{ij} = \inf I_{ij} \neq I_{ij} \\ ]-\infty, 0[& \text{when } I'_{ij} = \sup I_{ij} \neq I_{ij} \end{cases}$$

In [Dim02] we have shown that the family $(J_{ij})_{i,j}$ also satisfies the triangle property. J_{ij} gives the constraint for the difference between the number of δ^+'s and δ^-'s in between the i-th and the j-th action. Then, the saturated set of discretizations of L is the δ-language $\xi_1 a_1 \xi_2 a_2 \ldots a_{n-1} \xi_n$ in which ξ_i is a "shuffle" of $\underbrace{11\ldots1}_{\alpha_{i-1,i}}$ with a word $\zeta_i \in \{\delta^-, \delta^+\}^*$ with $|\zeta_i|_{\delta^+} - |\zeta_i|_{\delta^-} \in J_{i-1,i}$.

The following property is straightforward:

Proposition 2. *Given two saturated δ-languages $W_1, W_2 \subseteq \Delta^*$, if $\|W_1\| \cap \|W_2\| \neq \emptyset$ then $W_1 \cap W_2 \neq \emptyset$. Moreover, if $z \notin \Sigma$, then $W_1 z W_2$ is a saturated δ-language and*

$$\|W_1 z\| \cdot \|W_2\| = \|W_1 \cdot z \cdot W_2\|$$

Proof. The first part of the proposition is trivial, by definition of saturated sets. We will prove the second part for W_1 and W_2 being equivalence classes w.r.t. the semantic equivalence.

Take $w_1 \in W_1$ and $w_2 \in W_2$. For the left-to-right inclusion, consider $\sigma \in \|w_1 z\| \|w_2\|$ and denote L the region timed language to which σ belongs. By the discussion above on the construction of saturated sets, we may construct a δ-word w with $\|w\| = L$. It is then easy to decompose w as $w = w_1' z w_2'$, and σ as $\sigma = \sigma_1 z \sigma_2$, and to observe that $\sigma_i \in \|w_i\| \cap \|w_i'\| = \|w_i\| = \|w_i'\| = L$. The right-to-left inclusion is straightforward, as is the property of $W_1 z W_2$ being saturated. □

Remark 4. In general, concatenation of saturated δ-languages does not give a saturated δ-language. Consider, for example, that $D \subseteq \Delta^*$ is the saturated δ-language which discretizes $L = \|\langle \underline{\mathbf{t}} \rangle_{]0,1[}\|$. Then, in the saturated discretization of $L \cdot L$ we have the δ-word $1\delta^+\delta^-1$. But it is clear that this δ-word cannot be decomposed into two δ-words ω_1 and ω_2 such that $\|\omega_1\| = \|\omega_2\| = L$. Therefore, $D \cdot D$ does not contain $1\delta^+\delta^-1$.

On the other hand, the intersection of two nonsaturated δ-languages might be empty, even when the intersection of their semantics is nonempty. But our aim is to construct discretizations of timed languages by union, concatenation, star and intersection applied to some basic δ-languages. If we do not solve the noncompositionality of intersection w.r.t. nonsaturation, we might get an empty δ-language as a discretization of a nonempty timed language. In fact, without solving this problem, the nonarchimedian discretization is no better than the region discretization.

The solution is then to work with images of saturated δ-languages under *deletion morphisms*. A δ-language $D \subseteq \Delta^*$ is **weakly saturated** iff there exists a set of symbols Γ with $\Gamma \cap \Sigma = \emptyset$ and a saturated δ-language $D' \subseteq \left(\Gamma \cup \Sigma \cup \{1\delta^+, \delta^-\}\right)^*$ such that $D = [\Gamma \mapsto \varepsilon]D'$. Here, $[\Gamma \mapsto \varepsilon]$ is the morphism which deletes any occurrence of a symbol from Γ. The following proposition ensures the correctness of this approach:

Proposition 3. *Given two weakly saturated δ-languages W_1, W_2, we have that*

1. *If $\|W_1\| \cap \|W_2\| \neq \emptyset$ then $W_1 \cap W_2 \neq \emptyset$ and $W_1 \cap W_2$ is a weakly saturated language.*
2. *$\|W_1 \cdot W_2\| = \|W_1\| \cdot \|W_2\|$ and $W_1 \cdot W_2$ is a weakly saturated language.*

Proof. We will first prove the following lemma:

Lemma 1. *Symbol deletion in δ-words commutes with semantics, that is, for each $\omega \in \Delta^*$ and $a \in \Sigma$, $\|[a \mapsto \varepsilon]\omega\| = [a \mapsto \varepsilon]\|\omega\|$. Moreover, $\|[a \mapsto \varepsilon]\omega\|$ is a region timed language.*

Proof. We will prove the result for $\|\omega\|_a = 1$, the general case following from the straightforward fact that renaming commutes with semantics.

For the left-to-right inclusion, suppose that $a_k = a$ for some $1 \leq k \leq n$. What is then required to prove is that the set

$$R' = \big\{(t_1, \ldots, t_{k-1}, t_{k+1}, \ldots, t_n) \mid \forall 0 \leq i < j \leq n \text{ with } i, j \neq k, t_j - t_i \in I_{ij}, t_0 = 0\big\}$$

(which is an $(n-1)$-dimensional region!) has the property that

$$R' \subseteq \big\{(t_1, \ldots, t_{k-1}, t_{k+1}, \ldots, t_n) \mid \exists t_k \in \mathbb{R}_{\geq 0} \text{ s.t. } (t_1, \ldots, t_{k-1}, t_k, t_{k+1}, \ldots, t_n) \in \mathcal{U}(\|\omega\|)\big\}$$

where $\mathcal{U}(\|\omega\|)$ is the underlying region for the region language $\|\omega\|$.

But this is a well-known property for conjunctions of difference constraints satisfying the triangle property, see [Dim01] for a proof. The right-to-left proof is straightforward, as well as the second conclusion. □

Proof (of proposition 3, continued). Let us first observe that, if $\|W_1\| \cap \|W_2\| \neq \emptyset$ then $\|W_1\| \cap \|W_2\|$ is a region timed language. This follows easily from the above lemma, by observing that the projection of an n-dimensional region onto a k-dimensional subspace of ${\mathbb{R}_{\geq 0}}^n$ is a k-dimensional region.

We will prove the first part of the result for $\|W_1\| = \|W_2\|$, the general case following by decomposing each W_i into equivalence classes. We will in fact only give the proof for a special case, the general case following by a tedious generalization of our proof.

Suppose that $\sigma \in \|W_1\| \cap \|W_2\|$ has the property that $\sigma = (t_1 a_1)(t_2 a_2)(t_3, \varepsilon)$. Hence, for each $\omega \in W_i$, $\omega = \eta_1 a_1 \eta_2 a_2 \eta_3$, for some $\eta_i \in \{1, \delta^+, \delta^-\}^*$. Moreover, suppose that the saturated set $\overline{W}_1$ which witnesses the weakly saturation of W_1 is the δ-language discretizing the region language $L_1 = \big\{(u_1, a_1)(u_2, x)(u_3, a_2)(u_4, \varepsilon) \mid u_j - u_i \in I_{ij} \forall 0 \leq i < j \leq 3\big\}$, for some family of intervals $(I_{ij})_{i,j}$ for which we have $t_1 \in I_{01}$, $t_2 \in I_{03}$, $t_3 \in I_{04}$, $t_2 - t_1 \in I_{13}$, $t_3 - t_1 \in I_{14}$, $t_3 - t_2 \in I_{34}$. Hence, $W_1 = [x \mapsto \varepsilon]\overline{W}_1$.

Similarly, suppose that the saturated set $\overline{W}_2$ which witnesses the weakly saturation of W_2 is the δ-language discretizing the region language $L_2 = \big\{(u_1, a_1)(u_2, y)(u_3, a_2)(u_4, \varepsilon) \mid u_j - u_i \in J_{ij} \forall 0 \leq i < j \leq 3\big\}$, for some family of intervals $(J_{ij})_{i,j}$ for which we have similar properties related to t_1, t_2 and t_3. Hence, $W_2 = [y \mapsto \varepsilon]\overline{W}_2$. It then follows that $I_{ij} = J_{ij}$ for all $0 \leq i, j \leq 4$ and $i, j \neq 2$.

Assume further that I_{12} precedes J_{12} on the real axis (we denote this by $I_{12} \leq J_{12}$). Then the timed language

$$L_1 = \big\{(u_1, a_1)(u_2, x)(v_2, y)(u_3, a_2)(u_4, \varepsilon) \mid u_j - u_i \in I_{ij} \forall 0 \leq i < j \leq 3, \\ v_2 - u_i \in J_{i2} \forall 0 \leq i \leq 1, u_i - v_2 \in J_{2i} \forall 3 \leq i \leq 4\big\}$$

is a union of region languages, hence can discretized by a saturated set $\overline{W}$.

But $[x \mapsto \varepsilon]\overline{W} = \overline{W}_2$ and $[y \mapsto \varepsilon]\overline{W} = \overline{W}_1$. This implies that

$$W_1 \cap W_2 = \big([\{x, y\} \mapsto \varepsilon]\overline{W}$$

which shows that $W_1 \cap W_2$ is nonempty and is a weakly saturated set. The case when $J_{12} \geq I_{12}$ is treated similarly.

Let us not focus on the second property to be proved. Consider again $\overline{W}_1, \overline{W}_2$ saturated δ-languages with $W_1 = [x \mapsto \varepsilon]\overline{W}_1$ and $W_2 = [y \mapsto \varepsilon]\overline{W}_2$, with $x \neq y$. By taking a third different symbol $z \notin \Sigma$, we have that $W_1 \cdot W_2 = [\{x, y, z\} \mapsto \varepsilon](W_1 \cdot z \cdot W_2)$, hence $W_1 \cdot W_2$ is a weakly saturated δ-language.

Therefore, by applying lemma 1 and proposition 2,

$$\|W_1\| \cdot \|W_2\| = [\{x, y, z\} \mapsto \varepsilon]\|\overline{W}_1 \cdot z \cdot \overline{W}_2\| = \|W_1 \cdot W_2\| \qquad \square$$

It remains to identify a class of saturated languages over Δ^* that contains exactly the languages whose semantics equals the timed language of some timed automaton. This is the subject of the next section. But before that, we will show here that, in general, discretizations of languages of timed automata are context-sensitive languages over Δ. To see that, consider the following discretization of the (semantics of the) expression $\langle \underline{\mathbf{t}}a_1\underline{\mathbf{t}}a_2\rangle_{]2,3[}\underline{\mathbf{t}}a_3 \wedge \underline{\mathbf{t}}a_1\langle\underline{\mathbf{t}}a_2\underline{\mathbf{t}}a_3\rangle_{]1,2[}$:

$$D = \{\eta_1 a_1 \eta_2 a_2 \eta_3 a_3 \mid |\eta_1\eta_2|_{\delta^+} > |\eta_1\eta_2|_{\delta^-}, |\eta_2\eta_3|_{\delta^+} < |\eta_2\eta_3|_{\delta^-}, |\eta_1|_1 = |\eta_2|_1 = |\eta_3|_1 = 1\}$$

It is easy to see that this language is context-sensitive: its intersection with the regular language $(\delta^+)^*(\delta^-)^*1a_1(\delta^+)^*(\delta^-)^*1a_2(\delta^+)^*(\delta^-)^*1a_3$ is

$$L_4 = \left\{(\delta^+)^m(\delta^-)^n 1a_1(\delta^+)^p(\delta^-)^q 1a_2(\delta^+)^r(\delta^-)^s 1a_3 \mid m - n > p - q, p - q < r - s\right\}$$

and L_4 is an easy prey to the pumping lemma for context-free languages.

4 Oscillator automata

An appropriate class of automata for accepting saturated sets of δ-words must have the ability to compare numbers of δ^+'s and δ^-'s in infixes of δ-words, but, for decidability reasons, it must be unable to *count* them. The formalization of these leads to the notion of *oscillator automata*, that we present here.

Before that, let us give some notations used in this section. We will use the symbol $\forall$ as the symbol of the total binary relation on the integers, that is, $x\forall y$ for all $x, y \in \mathbb{Z}$. We denote $\mathcal{R} = \{<, =, >, \}$ and $\mathcal{R}_\forall = \mathcal{R} \cup \{\forall\}$. For any $n \in \mathbb{N}$, $R^n_\forall$ denotes the n-tuple $R^n_\forall = (\forall, \ldots, \forall)$. For $c \in \mathbb{Z}^n$ and $X \subseteq [1 \ldots n]$, we denote by $c[X := 0]$ the tuple whose components are $c[X := 0]_i = c_i$ for $i \notin X$ and $c[X := 0]_i = 0$ for $i \in X$. The unit tuple is denoted $\mathbf{1} = (1, \ldots, 1)$, and the zero tuple is denoted $\mathbf{0}$. We will also use addition and subtraction on tuples. Hence, $(c + \mathbf{1})_i = c_i + 1$ and $(c - \mathbf{1})_i = c_i - 1$ for all $i \in [1 \ldots n]$.

Definition 2. *An* ***oscillator automaton of degree*** n *(or* n***-oscillator automaton****) is a tuple* $\mathcal{A} = (Q, \theta, Q_0, Q_f)$ *where* Q *is a finite set of* states*,* $Q_0, Q_f \subseteq Q$ *are the sets of* initial*, resp.* final *states, while* $\theta \subseteq Q \times \Delta \times \mathcal{R}^n_\forall \times [1 \ldots n]Q$ *is the transition relation, where* $[1 \ldots n] = \{1, \ldots, n\}$*. Additionally, it is required that* $(q, \delta^+, R, q') \in \theta$ *if and only if* $(q, \delta^-, R, q') \in \theta$*, and in this case* $q = q'$ *and* $R = R^n_\forall$*.*

A *configuration* of an oscillator automaton is a tuple $(q, c_1, \ldots, c_n) \subseteq Q \times \mathbb{Z}^n$ consisting of a state of the automaton and an n-tuple of *counters*. We denote the set of configurations of $\mathcal{A}$ as $\mathcal{C}(\mathcal{A})$. The set of *initial configurations* is the set $Q_0 \times \{(0, 0, \ldots, 0)\}$, while the set of *final configurations* is the set $Q_f \times \mathbb{Z}^n$.

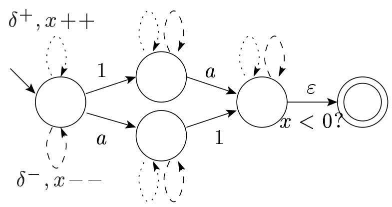
Transitions between configurations are constructed in the following way: for any $x \in \Delta$ for which $(q, x, R, X, q') \in \theta$, if we denote $X = \{i \in [1 \dots n] \mid R_i \neq \forall\}$, then $(q, c_1, \dots, c_n) \vdash^x (q', c'_1, \dots, c'_n)$ if for each $i \in [1 \dots n]$, $c_i R_i 0$ and

– If $x \notin \{\delta^-, \delta^+\}$ then $c' = c[X := 0]$.
– If $x = \delta^+$ then $c' = c + \mathbf{1}$, and if $x = \delta^-$ then $c' = c - \mathbf{1}$. Observe that, in this case, $R = R^n_\forall$.

Hence, in a tuple $(q, x, R, X, q') \in \theta$, the R component gives the *precondition* that the counters must meet in order to take the transition. In R, non-$\forall$-components give the subset of counters that are reset after taking the transition. Moreover, only δ^+-transitions increment counters, while only δ^--transitions decrement counters.

A *run* in $\mathcal{A}$ is a sequence of configurations $\rho = \big((q_i, c^i_1, \dots, c^i_n)\big)_{i \in [0 \dots k]}$ connected by transitions, that is, $(q_{i-1}, c^{i-1}_1, \dots, c^{i-1}_n) \vdash^{x_i} (q_i, c^i_1, \dots, c^i_n)$ for each $i \in [1 \dots n]$. We say that the δ-word $x_1 \dots x_n$ is *associated* to the run ρ. A run is *accepting* if it starts with an initial configuration and ends in a final configuration. The δ-language*accepted* by $\mathcal{A}$ is the set of δ-words $\omega = x_1 \dots x_k$ associated to an accepting run, and is denoted $D(\mathcal{A})$. The *timed language accepted* by $\mathcal{A}$, denoted $L(\mathcal{A})$, is $\|D(\mathcal{A})\|$

As an example, the 1-oscillator automaton below accepts the discretization of the timed language $\|\langle \underline{\mathbf{t}} a \underline{\mathbf{t}} \rangle_{]0,1[)}\|$. The dotted arrows represent the δ^+-transitions, while the dashed arrows represent the δ^--transitions.



Theorem 2. *The emptiness problem is decidable for oscillator automata.*

Proof. The proof idea is that, for checking emptiness, we only need to keep the the sign of each counter, together with the sign of the difference between each pair of counters. Essentially, this information is similar to the *ordering of the fractional parts* in the region construction [AD94], and will be called here as *pattern*.

An n-*relation pattern*, or simply an n-pattern, is a surjective function $\varphi : [0 \dots n] \to [1 \dots k]$, where $k \leq n + 1$. An n-pattern abstracts a tuple of n integer values $c = (c_1, \dots, c_n)$ to the *order* between the components of the tuple and the *sign* of these components. The relation pattern *abstracting* a tuple $c = (c_1, \dots, c_n)$ is the n-pattern $\mathsf{abs}_c : [0 \dots n] \to [1 \dots k]$ for which $\mathsf{abs}_c(i) = j$ if c_i the j-th largest value in the tuple $(0, c_1, \dots, c_n)$ (note that we have extended the tuple c with the value 0). For example, in the n-pattern abs_c abstracting the tuple $c = (-2, 1, -2, 0)$, $\mathsf{abs}_c(4) = 2$ since c_4 is the second largest value, while $\mathsf{abs}_c(0) = \mathsf{abs}_c(4) = 2$, $\mathsf{abs}_c(1) = \mathsf{abs}_c(3) = 1$ and $\mathsf{abs}_c(2) = 2$. The *zero n-pattern* is the unique function $\varphi_0 : [0 \dots n] \to \{1\}$. A tuple $R \in \mathcal{R}^n_\forall$ is said *compatible* with a n-pattern φ iff, for all $i \in [1 \dots n]$, $\varphi(i) R_i \varphi(0)$.

The transitions between configurations of $\mathcal{A}$ have three effects on the tuples of integers: they may increment or decrement tuples, or reset certain components of these to zero. We need to define appropriate operations on n-patterns that abstract these three operations. To this end, we associate to each n-pattern φ the following three mappings: $v_1(\varphi) : [0 \dots n] \to [1 \dots k - 1]$, $v_2(\varphi) : [0 \dots n] \to [1 \dots k]$ and $v_3(\varphi) : [0 \dots n] \to [1 \dots k + 1]$, defined as follows:

1. For each $i \in [1 \ldots n]$ with $\varphi(i) < \varphi(0)$, $v_1(\varphi)(i) = v_2(\varphi)(i) = v_3(\varphi)(i) = \varphi(i)$.
2. $v_1(\varphi)(0) = v_2(\varphi)(0) = \varphi(0) - 1$ and $v_3(\varphi)(0) = \varphi(0)$.
3. For each $i \in [1 \ldots n]$ with $\varphi(i) \geq \varphi(0)$, $v_1(\varphi)(i) = \varphi(i) - 1$, $v_2(\varphi)(i) = \varphi(i)$ and $v_3(\varphi)(i) = \varphi(i) + 1$.

Let us observe that the three above mappings are not always n-patterns: $v_1(\varphi)$ is a n-pattern iff $\varphi(0) \geq 2$, $v_2(\varphi)$ is a n-pattern iff $\varphi(0) \geq 2$ and $card(\varphi^{-1}(\varphi(0))) \geq 2$, while $v_3(\varphi)$ is a n-pattern iff $card(\varphi^{-1}(\varphi(0))) \geq 2$.

The *increment* of φ is the set of n-patterns

$$\varphi^{++} = \begin{cases} \{\varphi\} \cup \{v_1(\varphi) \mid \varphi(0) \geq 2\} & \text{iff } card(\varphi^{-1}(\varphi(0))) = 1 \\ \{v_3(\varphi)\} \cup \{v_2(\varphi) \mid \varphi(0) \geq 2\} & \text{iff } card(\varphi^{-1}(\varphi(0))) \geq 2 \end{cases}$$

Let us make more clear the connection between increments on n-patterns and increments on n-tuples: the incrementation of the tuple $c_1 = (-2, 1, -2, 0)$ gives the tuple $c_1' = (-1, 2, -1, 1)$, while the incrementation of the tuple $c_2 = (-1, 1, -1, 0)$ gives the tuple $c_2' = (0, 2, 0, 1)$. Both c_1 and c_2 are abstracted to the same n-pattern. However it is clear that c_1' and c_2' are not abstracted to the same n-pattern. In fact, $\mathsf{abs}_{c_1'} = v_3(\mathsf{abs}_c)$ and $\mathsf{abs}_{c_2'} = v_2(\mathsf{abs}_c)$.

Similarly, we may define the *decrement* of φ, as the set

$$\varphi^{--} = \begin{cases} \{\varphi\} \cup \{v_1'(\varphi) \mid \varphi(0) \geq 2\} & \text{iff } card(\varphi^{-1}(\varphi(0))) = 1 \\ \{v_3'(\varphi)\} \cup \{v_2'(\varphi) \mid \varphi(0) \geq 2\} & \text{iff } card(\varphi^{-1}(\varphi(0))) \geq 2 \end{cases}$$

where the n-patterns $v_1'(\varphi) : [0 \ldots n] \to [1 \ldots k-1]$, $v_2'(\varphi) : [0 \ldots n] \to [1 \ldots k]$ and $v_3'(\varphi) : [0 \ldots n] \to [1 \ldots k+1]$ are defined as follows:

1. For each $i \in [1 \ldots n]$ with $\varphi(i) \leq \varphi(0)$, $v_1'(\varphi)(i) = v_2'(\varphi)(i) = v_3'(\varphi)(i) = \varphi(i)$.
2. $v_1'(\varphi)(0) = \varphi(0)$ and $v_2'(\varphi)(0) = v_3'(\varphi)(0) = \varphi(0) + 1$.
3. For each $i \in [1 \ldots n]$ with $\varphi(i) > \varphi(0)$, $v_1'(\varphi)(i) = \varphi(i) - 1$, $v_2'(\varphi)(i) = \varphi(i)$ and $v_3'(\varphi)(i) = \varphi(i) + 1$.

Finally, given a subset $X \subseteq [1 \ldots n]$, the X-*reset* of a n-pattern φ is the n-pattern $\varphi[X := 0]$ defined as follows:

1. $\varphi[X := 0] = card\{\varphi(i) \mid \varphi(i) \leq \varphi(0), i \notin X\}$.
2. For each $i \in X$, $\varphi[X := 0](i) = \varphi[X := 0](0)$.
3. For each $i \in X$, $\varphi[X := 0](i) = card\{\varphi(j) \mid \varphi(j) \leq \varphi(i), j \notin X\}$.

We then construct the *finite* automaton $\overline{\mathcal{A}} = (\overline{Q}, \overline{\theta}, \overline{Q}_0, \overline{Q}_f)$ where $\overline{Q} = Q \times \mathsf{Pat}_n$, $\overline{Q}_0 = Q_0 \times \{\varphi_0\}$, $\overline{Q}_f = Q_f \times \mathsf{Pat}_n$ and

$$\begin{aligned} \overline{\theta} = \big\{(q,\varphi) \xrightarrow{\delta+} (q,\varphi') \mid \varphi' \in \varphi^{++}\big\} \cup \big\{(q,\varphi) \xrightarrow{\delta-} (q,\varphi') \mid \varphi' \in \varphi^{--}\big\} \cup \\ \big\{(q,\varphi) \xrightarrow{x} (q',\varphi') \mid x \in \Sigma \cup \{1\} \text{ and } \exists (q,x,R,X,q') \in \theta \text{ s.t.} \\ \varphi \text{ is compatible with } R, X = \{i \in [1 \ldots n] \mid R_i \neq \forall\} \text{ and } \varphi' = \varphi[X := 0]\big\} \end{aligned}$$

It is easy to observe that, if the oscillator automaton $\mathcal{A}$ has a nonempty language, then the associated finite automaton $\overline{\mathcal{A}}$ has a nonempty language too. The proof of the reverse implication is given in the appendix. □

Note that the first requirement in the definition of oscillator automata – the requirement that the increment and decrement transitions do not change the state of the automaton – is essential in the proof of the decidability theorem. Without this requirement, it would be easy to code Minsky machines [HU92] with increment and decrement steps.

An upper bound for the number of states in the finite automaton $\overline{\mathcal{A}}$ is $n! \cdot 2^n \cdot card(Q)$.

5 From timed automata to oscillator automata and back

In this section we show that oscillator automata represent exactly all discretizations of languages of timed automata. The proof of the right-to-left inclusion is based upon the Kleene theorem 1, while the proof in the other direction actually constructs a region automaton, viewed as a timed automaton.

Theorem 3. *For each timed regular expression E there exists an oscillator automaton whose accepted timed language is $\|E\|$.*

Proof. The idea is to recursively construct *weakly saturated* oscillator automata for the subexpressions of E, by showing that union, intersection, concatenation, star and renaming preserves the property of being weakly saturated.

For the expression $\underline{\mathbf{t}}z$, with $z \in \Sigma \cup \{\varepsilon\}$, the oscillator automaton has two states q_1 and q_2, with q_1 initial and q_2 final, and it loops in q_1 with clock ticks and delay symbols. Then, on a z symbol, it moves to q_2, which has no outgoing transitions. Observe that this automaton is actually saturated.

For the expression $\langle E \rangle_I$, observe first that $\|\langle E \rangle_I\| = \left\| E \wedge \left\langle \underline{\mathbf{t}} + \sum_{a \in \Sigma} a \right\rangle_I \right\|$. Hence we only need to construct oscillator automata for expressions of the type $\left\langle \underline{\mathbf{t}} + \sum_{a \in \Sigma} a \right\rangle_I$. We do this for the case of $I =]c, c+1[$ for some $c \in \mathbb{N}$, the case with $I = \{c\}$ being similar, and the general case following from these by decomposing I into point or open unit length intervals.

For the chosen case, we may construct a saturated 1-oscillator automaton that has $2c + 4$ states $Q = \{q_0, \ldots, q_{c+1}, r_1, \ldots, r_{c+2}\}$ with q_0 initial and q_{c+1}, r_{c+2} final. It loops in any state with any symbol $x \neq 1$, and moves from q_{i-1} to q_i on symbol 1, where $i \in [2 \ldots c]$. It also moves from q_0 to r_1 and from r_{i-1} to r_i on symbol 1, with $i \in [2 \ldots c+1]$. Finally, an ε-transition moves the automaton from q_c to q_{c+1}, and this transition is guarded by the relation $>$. Also, another ε-transition moves the automaton from q_c to q_{c+1}, and this transition is guarded by the relation $<$. Again, the resulting oscillator automaton is saturated.

Suppose now we have two oscillator automata $\mathcal{A}_1$ and $\mathcal{A}_2$ with weakly saturated languages. We may suppose, without loss of generality, for both $i = 1, 2$, that $\mathcal{A}_i$ is obtained from a saturated oscillator automaton $\overline{\mathcal{A}}_i$ by replacing, on all transitions, all labels not in Σ with ε. That is, $\mathcal{A}_i$ and $\overline{\mathcal{A}}_i$ have the same state space and

- For all $a \in \Sigma \cup \{1, \delta^+, \delta^-\}$, $(q, a, R, X, q') \in \delta_{\mathcal{A}_i}$ iff $(q, a, R, X, q') \in \delta_{\overline{\mathcal{A}}_i}$
- For all $x \notin \Sigma$, if $(q, x, R, X, q') \in \delta_{\overline{\mathcal{A}}_i}$ then $(q, \varepsilon, R, X, q') \in \delta_{\mathcal{A}_i}$,
- If $(q, \varepsilon, R, q') \in \delta_{\mathcal{A}_i}$ and q is not initial and q' not final, then there exists $x \notin \Sigma$ such that $(q, x, R, X, q') \in \delta_{\overline{\mathcal{A}}_i}$.

We will also suppose that, in both $\mathcal{A}_1$ and $\mathcal{A}_2$, the only transitions leaving initial states are either δ^+-loops, or δ^--loops, or ε-transitions with constraint $R^n_\forall$. Similarly, we suppose that, in both $\mathcal{A}_1$ and $\mathcal{A}_2$, the only transitions entering final states are either δ^+-loops, or δ^--loops, or ε-transitions with constraint $R^n_\forall$. It is easy to transform a saturated oscillator automaton in order to obtain these properties.

We may then build easily a saturated oscillator automaton $\mathcal{B}$ for $D(\overline{\mathcal{A}}_1) \cdot z \cdot D(\overline{\mathcal{A}}_2)$, by simply connecting each final state q of $\overline{\mathcal{A}}_1$ to each initial state q' of $\overline{\mathcal{A}}_2$ via a z-transition with constraint $R^n_\forall$, where z is a fresh symbol, not used in any of the automata. By proposition 2, $D(\mathcal{B})$ is saturated. Then, by proposition 3, if we replace in $\mathcal{B}$ all labels not in Σ with ε, the resulting automaton accepts $D(\mathcal{A}_1) \cdot D(\mathcal{A}_2)$. It is also clear that this automaton is weakly saturated, as it's δ-language is $[\Sigma_1 \cup \Sigma_2 \cup \{z\} \mapsto \varepsilon]D(\mathcal{B})$.

A similar construction works for $D(\mathcal{A}_1)^+$: we draw z-transitions from each final state of $\mathcal{A}_1$ to each of its initial state, with constraint $R^n_\forall$. Coping with $D(\mathcal{A}_1)^*$ is then trivial: we only need to append a small saturated automaton discretizing exactly the timed language containing only the empty timed word.

A straightforward union construction for $\overline{\mathcal{A}}_1$ and $\overline{\mathcal{A}}_2$ assures also that $D(\mathcal{A}_1) \cup D(\mathcal{A}_2)$ is accepted by a weakly nonsaturated oscillator automaton. Also, for some $a \in \Sigma$ and $b \in \Sigma \cup \{\varepsilon\}$, the δ-language $[a \mapsto b]D(\mathcal{A}_1)$ is accepted by the oscillator automaton which results from $\mathcal{A}_1$ by replacing a labels with b labels on all a-transitions.

The remaining case is intersection. For this operation, we need to modify $\overline{\mathcal{A}}_1$ by appending, in each non-initial non-final state, loops with all symbols in Σ_2, with constraint $R^n_\forall$. Denote $\tilde{\mathcal{A}}_1$ the resulting automaton. Similarly, we put loops in each state of $\overline{\mathcal{A}}_2$ with all symbols in Σ_1 and modify initial and final states accordingly, and denote $\tilde{\mathcal{A}}_2$ the resulting automaton. Hence, $D(\mathcal{A}_i) = [\Sigma_1 \cup \Sigma_2]D(\tilde{\mathcal{A}}_i)$, $i = 1, 2$. Also note that both $\tilde{\mathcal{A}}_1$ and $\tilde{\mathcal{A}}_2$ are saturated. Then the classical intersection construction will provide a saturated oscillator automaton $\tilde{\mathcal{A}}$ for $D(\tilde{\mathcal{A}}_1) \cap D(\tilde{\mathcal{A}}_2)$, from which, by deleting all symbols in $\Sigma_1 \cup \Sigma_2$, we get a weakly saturated oscillator automaton for $D(\mathcal{A}_1) \cap D(\mathcal{A}_2)$. The automaton $\tilde{\mathcal{A}}$ is $\tilde{\mathcal{A}} = (Q_1 \times Q_2, \tilde{\delta}, Q_0^1, Q_0^2, Q_f^1, Q_f^2)$ where

$$\begin{aligned}\tilde{\delta} = \big\{(q_1, q_2, x, R, X, q_1', q_2') \mid \exists (q_1, x, R_1, X_1, q_1') \in \tilde{\delta}_1, (q_2, x, R_2, X_2, q_2') \in \tilde{\delta}_2,& \\ X = X_1 \cup X_2, R = R_1 \times R_2\big\}&\end{aligned}$$

Here, $\tilde{\delta}_i$ is the transition relation in $\tilde{\mathcal{A}}_i$, $i = 1, 2$. □

Theorem 4. *For each oscillator automaton $\mathcal{A}$ there exists a timed automaton accepting $L(\mathcal{A})$.*

Proof. Suppose that $\mathcal{A}$ has n counters. For each counter c_i of $\mathcal{A}$ we will use a clock y_i in the timed automaton. Moreover, we will employ a new clock y_0, which will be used for marking clock ticks. Hence, almost each 1-transition in $\mathcal{A}$ will give a transition in which we check $y_0 = 1$ and reset y_0. The only transitions which will not be transformed into such clock tick transitions will be the first 1-transitions. On each accepted run, the first 1-transition is either ignored, or is transformed into a $(y_0 = 1?, y_0 := 0)$ transition too.

Before going any further, let us note that, with this simple transformation of $\mathcal{A}$ into a timed automaton $\mathcal{B}$ (that is, even without any other clock constraint), on any accepting run in $\mathcal{B}$ we have the property that the interval separating any pair of actions a, b, is $]n - 1, n + 1[$, where n is the number of clock ticks on the same run, but in $\mathcal{A}$. This is insufficient if we have runs in which a counter c_i is reset on a and tested only to be positive on b. This would mean that the amount of time elapse between a and b is in the interval $]n, n + 1[$, but not in $]n - 1, n[$. We therefore have to use the test on c_i on the b-transition for generating a constraint on the corresponding clock y_i which eliminates the part $]n-1,n]$ from the interval separating a and b.

To this end, for each pair of states $q, q' \in Q$ we put in a set $N(q,q')$ all the numbers of 1s occurring on a sequence of $\mathcal{A}$-transitions between q and q'. Note that we do not need to check whether that sequence of transitions is feasible. The computation of $N(q,q')$ is a matter of regular sets. We then separate $N(q,q')$ into two sets: $N^0(q,q')$, containing even numbers, and $N^1(q,q')$, with odd numbers. Then, for each $1 \le i \le n$ and on each transition $\tau_1 = (q,x,R,X,r)$ on which counter c_i is reset, we start counting modulo 2 the 1s into a register h_i.

Take a subsequent transition $\tau_2 = (q',x',R',X',r')$ on which R_i is $<$. This transition is transformed in $\mathcal{B}$ into a transition with constraint $y_i \in]N^{h_i}(q,r')-1, N^{h_i}(q,r')[$. When R_i is $=$ we put the constraint $y_i = N^{h_i}(q,r')$, and similarly for R_i being $>$. To see how this solves our problem, denote n_i the number of 1s encountered since c_i was reset. Obviously $n \in N^{h_i}(q,r')$. Therefore, if no transition leaving q' has $R_i =$ ‘ $>$ ‘ then the constraints we have designed assure that the interval between passing through τ_1 and τ_2 has an empty intersection with $]n-1,n[$.

The formalization of this is the following: denote $\mathcal{A} = (Q, \Sigma, \delta, Q_0, Q_f)$ the oscillator automaton, and $\mathcal{Y} = \{y_1,\ldots,y_n\}$. The timed automaton $\mathcal{B}$ has tuples $(q,\overline{q},p,h) \in Q \times Q^n \times \{-,+\} \times \{0,1\}^n$ as states, where p tells whether the first 1-transition has been encountered or not, $\overline{q}_i$ remembers the last state where the counter c_i was reset, while h_i are the registers modulo 2 whose use was shown above.

The transitions in $\mathcal{B}$ are the defined in the following form:

1. For each $(q,x,R,X,q') \in \delta$ with $x \in \Sigma \cup \varepsilon$ we have a transition $(q,\overline{q},p,h) \xrightarrow{a,C,Y} (q',\overline{q}',p,h')$ where $Y \subseteq \mathcal{Y}$ and C is a conjunction of constraints, both defined as follows:
   (a) For each $1 \le i \le n$, $y_i \in Y$ iff $c_i \in X$ and in this case $h'_i = 0$ and $\overline{q}'_i = q$; otherwise $h'_i = h_i$ and $\overline{q}'_i = \overline{q}_i$.
   (b) C contains $y_0 \in]0,1[$ as a conjunct;
   (c) For each $1 \le i \le n$ for which R_i is $<$, C contains $y_i \in]N^{h_i}(\overline{q}_i,q')-1, N^{h_i}(\overline{q}_i,q')[$.
   (d) For each $1 \le i \le n$ for which R_i is $=$, C contains $y_i = N^{h_i}(\overline{q}_i,q')$.
   (e) For each $1 \le i \le n$ for which R_i is $>$, C contains $y_i \in]N^{h_i}(\overline{q}_i,q'), N^{h_i}(\overline{q}_i,q')+1[$.
2. For each $(q,1,R,X,q') \in \delta$ we have a transition $(q,\overline{q},p,h) \xrightarrow{\varepsilon,C,Y} (q',\overline{q}',p',h')$ where $Y \subseteq \mathcal{Y}$ and C is a conjunction of constraints, both satisfying the requirement 1a from the previous point, as well as the following:
   (f) For each $1 \le i \le n$ for which R_i is $<$, C contains $y_i \in]N^{h_i+1}(\overline{q}_i,q')-1, N^{h_i}(\overline{q}_i,q')[$.
   (g) For each $1 \le i \le n$ for which R_i is $=$, C contains $y_i = N^{h_i+1}(\overline{q}_i,q')$.
   (h) For each $1 \le i \le n$ for which R_i is $>$, C contains $y_i \in]N^{h_i+1}(\overline{q}_i,q'), N^{h_i}(\overline{q}_i,q')+1[$.
   (i) If p is $+$ then $p' = p$, C contains the conjunct $c_0 = 1$ and $c_0 \in Y$; otherwise p' is $+$.
   (j) For each $1 \le i \le n$, if $c_i \notin X$ then $h'_i = (h_i+1) \mod 2$.

The δ^+ and δ^- transitions of $\mathcal{A}$ do not generate transitions in $\mathcal{B}$. The initial states of $\mathcal{B}$ are tuples $(q,\overline{q},p,h)$ with $q \in Q_0$, $\overline{q}_i = q_0$, $h_i = 0$ and p being $-$. The final states of $\mathcal{B}$ are tuples $(q,\overline{q},p,h)$ with $q \in Q_f$. □

6 Conclusions

We have presented a new discretization technique for timed automata, which corrects the non-compositionality of the region construction w.r.t. intersection. The discretization we have proposed makes use of two delay symbols with “nonarchimedian” semantics. Our discretizations are words over an alphabet containing the action symbols, clock tick symbols and the delay symbols. We have also investigated a class of automata that accepts discretizations of languages of timed automata.

The significance for model-checking is yet unclear. As noted at the end of section 4, the size of the finite automaton built for checking whether an n-oscillator automaton has an empty language is at most $n! \cdot 2^n \cdot card(Q)$. Comparing with the region construction, we observe that the largest constants used within clock constraints are hidden within $card(Q)$. On the other hand, the common basis for comparison should be the emptiness checking for timed regular expressions. But timed regular expressions are a very inefficient specification language, since, by [ACM02] in order to specify the language of a timed automaton, its untimed behavior is specified n times, once for each clock. More research is also needed in order to check whether compositional model checking or test generation may benefit from our approach.

Oscillator automata can be used on their own as a model for timed systems. We draw the attention also to the special form of timed automata which are built from oscillator automata. It is easy to check, for those automata, whether the accepted language is k-bounded, and hence whether the automaton is determinizable and complementable.

We believe that our approach may help in importing more results from classical language theory. The first subject that may take advantage of our approach is the adaptation of results concerning monoid recognizability. Our discretization might also give an alternative framework for laying the basis of a timed trace theory [DT99].

References

[ACM02] E. Asarin, P. Caspi, and O. Maler. Timed regular expressions. *Journal of ACM*, 49:172–206, 2002.

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AD03] E. Asarin and C. Dima. Balanced timed regular expressions. *ENTCS*, 68, issue 5, 2003.

[Bea98] D. Beauquier. Pumping lemmas for timed automata. In *Proceedings of FoSSaCS’98*, volume 1378 of *LNCS*, pages 81–94, 1998.

[Bel57] R. Bellmann. *Dynamic Programming*. Princeton University Press, 1957.

[BP99] P. Bouyer and A. Petit. Decomposition and composition of timed automata. In *Proceedings of ICALP’99*, volume 1644 of *LNCS*, pages 210–219, 1999.

[BP02] P. Bouyer and A. Petit. A Kleene/Büchi-like theorem for clock languages. *Journal of Automata, Languages and Combinatorics*, 2002. To appear.

[BPT01] P. Bouyer, A. Petit, and D. Thrien. An algebraic characterization of data and timed languages. In *Proceedings of CONCUR’2001*, volume 2154 of *LNCS*, pages 248–261, 2001.

[Dim01] C. Dima. *An algebraic theory of real-time formal languages*. PhD thesis, Université Joseph Fourier Grenoble, France, 2001.

[Dim02] C. Dima. Computing reachability relations in timed automata. In *Proceedings of LICS’02*, pages 177–186, 2002.

[DT99] D. D’Souza and P. S. Thiagarajan. Product interval automata: A subclass of timed automata. In *Proceedings of FSTTCS’99*, volume 1738 of *LNCS*, pages 60–71, 1999.

[Her98] Ph. Herrmann. Timed automata and recognizability. *Information Processing Letters*, 65:313–318, 1998.

[HU92] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley/Narosa Publishing House, 1992.

[LPY97] K. G. Larsen, Paul Petterson, and Wang Yi. Uppaal: Status & developments. In *Proceedings of CAV’97*, LNCS, pages 456–459, 1997.

[Yov98] S. Yovine. Model-checking timed automata. In *Lectures on Embedded Systems*, volume 1494 of *LNCS*, pages 114–152, 1998.

Appendix: Proof of Proposition 1

Let us denote I the $(k+2) \times (k+2)$ matrix of intervals which occurs in the definition of the semantics of a δ-word $\|\omega\|$, that is, for each $1 \leq i \leq j \leq k+2$,

$$I_{ij} = \begin{cases}]v_{ij}, v_{ij}+1[& \text{when } p_{ij} > n_{ij}, \\ \{v_{ij}\} & \text{when } p_{ij} = n_{ij}, \\ ]v_{ij}-1, v_{ij}[& \text{when } p_{ij} < n_{ij}, \end{cases}$$

and putting $I_{ji} = -I_{ij}$ (that is, if $I_{ij} =]2,3[$ then $I_{ji} =]-3,-2[$).

Such matrices of intervals are similar to difference-bound matrices [Bel57] and were studied in [Dim02]. The matrix I represents the set of $(k+2)$-tuples of reals $(x_1, \ldots, x_{k+2})$ for which $x_j - x_i \in I_{ij}$.

A sufficient condition for a matrix I to represent a non-empty set of tuples is the following [Dim02]: for all $i, j, l \in [1 \ldots k+2]$, $I_{ij} + I_{jl} \supseteq I_{il}$. Here real addition is extended over sets in the straightforward way: $I_{ij} + I_{jl} = \{\alpha + \beta \mid \alpha \in I_{ij}, \beta \in I_{jl}\}$.

We then claim that the matrix I associated to any δ-word ω satisfies this condition. The proof goes by induction, and the induction step follows by a case analysis. One case for the induction step involves showing that $I_{i.k+2} + I_{k+2,j} \supseteq I_{ij}$, where $i, j \in [1 \ldots k+1]$. Let us assume, for example, that $I_{i,k+2} =]v_{i,k+2}, v_{i,k+2}+1[$, $I_{k+1,j} = \{-v_{j,k+2}\}$, and that $i \leq j$. This implies that $p_{i,k+2} > n_{i,k+2}$ and $p_{j,k+2} = n_{j,k+2}$. But $p_{i,k+2} = p_{ij} + p_{j,k+2}$, $n_{i,k+2} = n_{ij} + n_{j,k+2}$, hence we must have $p_{j,k+2} > n_{j,k+2}$, and hence, $I_{j,k+2} =]v_{j,k+2}, v_{j,k+2}+1[$.

On the other hand, $v_{i,k+2} = v_{ij} + v_{j,k+2}$. Hence, $I_{i,k+2} + I_{k+2,j} = I_{ij}$. The other cases can be proved similarly.

Consider a tuple $\overline{x} = (x_1, \ldots, x_{k+2})$ that is represented by the matrix I. We may choose $x_1 = 0$, since the tuple $(x_1 - x_1, \ldots, x_{k+2} - x_1)$ is still represented by I. We may then consider the $k+1$ tuple of sums $\overline{t} = (t_1, \ldots, t_{k+1})$ with $t_i = x_{i+1} - x_i$ for all $i \in [1 \ldots k+1]$. If all components of this tuple are nonnegative, then $\overline{t}$ generates a timed word $w = t_1 a_1 t_2 \ldots a_k t_{k+1}$. The timed word w is clearly in the semantics of ω, since its timing components verify the definition 1.

Hence, all that remains to be shown is that whenever ω satisfies the requirement $(*)$, then the tuple $\overline{t}$ has only nonnegative components. But this is straightforward, since the requirement $(*)$ implies that $I_{i,i+1} \subseteq [0, \infty[$ for all $i \in [1 \ldots k+1]$. $\square$

Proof of Theorem 2, the reverse implication

For the reverse implication, observe first that, if $(q,c) \vdash^x (q',c')$ is a transition between two configurations, then the configuration $(q', 2c')$ is reachable from $(q, 2c)$. This is straightforward when $x \notin \{\delta^-, \delta^+\}$, since we will then have $(q,2c) \vdash^x (q',2c')$. In the case of a δ^+transition (in which case $q = q'$ and $c' = c + \mathbf{1}$), we need to insert an intermediary configuration, that is, we will have $(q,2c) \vdash^{\delta^+} (q, 2c+\mathbf{1}) \vdash^{\delta^+} (q, 2c + 2 \cdot \mathbf{1} = 2c')$. We may then conclude the following:

Remark 5. If (q',c') is reachable from (q,c), then $(q',2c')$ is reachable from $(q,2c)$, by induction on the length of the run that connects (q,c) to (q',c').

We will then prove the following claim:

Claim. If, in $\overline{\mathcal{A}}$, (q', φ') is reachable from (q, φ), then there exists $c, c' \in \mathbb{Z}$ such that $\mathsf{abs}_c = \varphi$, $\mathsf{abs}_{c'} = \varphi'$ and (q', c') is reachable from (q, c) in $\mathcal{A}$.

This claim will be proved by induction on the length of the run that connects (q, φ) to (q', φ').

The base case, with runs of length one, is trivial. Suppose then that we have proved the claim for runs of length strictly smaller than k. Take then a run of length k, $\overline{\rho} = \big((q_i, \varphi_i)\big)_{i \in [0 \ldots k]}$. By induction, we have that there exists $c, c' \in \mathbb{Z}^n$ such that $\mathsf{abs}_c = \varphi_0$, $\mathsf{abs}_{c'} = \varphi_{k-1}$ and (q_{k-1}, c') is reachable from (q_0, c). According to the type of the transition $(q_{k-1}, \varphi_{k-1}) \vdash (q_k, \varphi_k)$, we have three cases:

1. $\varphi_k = \varphi_{k-1}$, case in which the claim is trivially verified.
2. $\varphi_k = \varphi_{k-1}[X := 0]$, for some $X \subseteq [1 \ldots n]$.
   But then, by choosing $c'' = c'[X := 0]$, it is easy to observe that $\mathsf{abs}_{c''} = \varphi_k$ and that $(q_{k-1}, c') \vdash (q_k, c'')$, fact which proves our claim in this case.
3. $\varphi_k \in \varphi_{k-1}^{++}$, or $\varphi_k \in \varphi_{k-1}^{--}$.

We will prove the claim only for incrementation, the other case being similar. Let us observe that, in this case, $q_{k-1} = q_k$, by construction of $\overline{\mathcal{A}}$.

The first impulse is to choose $c'' = c' + \mathbf{1}$ and to say that $(q_{k-1}, c') \vdash^{\delta^+} (q_k, c'')$. However, in general, if $\mathsf{abs}_{c'} = \varphi_{k-1}$ and $\varphi_k \in \varphi_{k-1}^{++}$, it does not follow that $\mathsf{abs}_{c''} = \varphi_k$, for this choice of c''. This was already observed on page 10.

Formally, there are two situations in which $\mathsf{abs}_{c''} \neq \varphi_k$:

1. When $\varphi_k = v_1(\varphi_{k-1})$ or $\varphi_k = v_2(\varphi_{k-1})$, and for all $i \in \varphi_{k-1}^{-1}(\varphi_{k-1}(0) - 1)$, $c_i \leq -2$. In this situation, $c''_i = c_i + 1 \neq 0$, but $\varphi_k(i) = \varphi_k(0)$, hence $\mathsf{abs}_{c''} \neq \varphi_k$.
   The solution here is to choose $c'' = c' - c_i \cdot \mathbf{1}$ (rather than $c'' = c' + \mathbf{1}$), since then we would have that (q_{k-1}, c'') is reachable in c_i increment steps from (q_{k-1}, c') and that $\mathsf{abs}_{c''} = \varphi_k$, which would prove the claim.
2. When $\varphi_k = v_3(\varphi_k)$ and for all $i \varphi_{k-1}^{-1}(\varphi_{k-1}(0) - 1)$, $c_i = -1$. In this situation, $c''_i = c_i + 1 = 0$, but $\varphi_k(i) = \varphi_k(0) - 1$, hence again $\mathsf{abs}_{c''} \neq \varphi_k$.
   The solution here is to replace both c and c' with their "double", that is, to work with $(q_0, 2c)$ and $(q_{k-1}, 2c')$, which can be connected via a run in $\mathcal{A}$ by the Remark 5. Replacing c' with $2c'$ would then imply that $2c'$ falls no longer in this second case.

This ends the proof of our claim.

The proof of the decidability result then follows easily: assuming $\overline{\mathcal{A}}$ has a nonempty language, we take some run $\overline{\rho}$ that starts in $(q_0, \varphi_0) \in \overline{Q}_0$ and ends in $(q, \varphi) \in \overline{Q}_f$. Then, by applying the proven claim, we get that there exist $c, c' \in \mathbb{Z}^n$ such that $\mathsf{abs}_c = \varphi_0$ and $\mathsf{abs}_{c'} = \varphi$, and (q, c') is reachable from (q_0, c) in $\mathcal{A}$. But we then only have to observe that the only n-tuple which is abstracted by the n-pattern φ_0 is the identically null tuple, which proves that (q_0, c) is an initial configuration in $\mathcal{A}$, while (q, c') is clearly a final configuration in $\mathcal{A}$. □