A study on shuffle, stopwatches and independently evolving clocks*

Cătălin Dima[1] and Ruggero Lanotte[2]

[1] LACL, Université Paris 12, 61 av. du Général de Gaulle, 94010 Créteil Cedex, France
[2] Università dell'Insubria, Via Carloni 78, 22100, Como, Italy

Abstract. We show that stopwatch automata are equivalent with timed shuffle expressions, an extension of timed regular expressions with the shuffle operation. Since the emptiness problem is undecidable for stopwatch automata, and hence also for timed shuffle expressions, we introduce a decidable subclass of stopwatch automata called partitioned stopwatch automata. We give for this class an equivalent subclass of timed shuffle expressions and investigate closure properties by showing that partitioned stopwatch automata are closed under union, concatenation, star, shuffle and renaming, but not under intersection. We also show that partitioned stopwatch automata are equivalent with distributed time-asynchronous automata, which are asynchronous compositions of timed automata in which time may evolve independently.

1 Introduction

Timed automata [AD94] and their stopwatch extensions [HKPV98] are a widely accepted and powerful model of real time systems. They are designed to model the interaction between continuous processes and discrete logic by means of continuous time variables called clocks or stopwatches. They were designed with the aim to translate to the real-time setting as much as possible from the classical automata/logic duality, which is one of the central pillars in the model-checking approach to formal verification [CGP99,BK08]. For example, logical characterizations of timed languages are studied in [HRS98,Wil94], regular expressions in [ACM02,BP99] and monoidal characterizations in [BPT03].

Timed automata models are applied in schedulability analysis [AM01,FMPY06,AAM06], providing some interesting new results which improve classical worst-case analysis techniques. It has first been observed in [AM02] that modeling preemptive scheduling may require the use of stopwatches, and thus the algorithmic analysis of preemptive scheduling policies is somewhat related with the construction of decidable subclasses of stopwatch automata. Preemptive scheduling can also be expressed with regular expressions endowed with a shuffle operator, as suggested in our previous paper [Dim05]. A third possibility consists of networks of "time-asynchronous" timed automata with independent time passage, first suggested in [Kri99]. Finally, a fourth approach is explored in [FKPY07], where the authors use timed automata with subtraction.

Our present study explores the connections between the first three above possibilities for modeling preemption and/or time independence. We focus on the comparison between the expressive power of stopwatches, shuffle and independently evolving clocks. We also study closure properties of the classes of languages constructed using the different formalisms.

We first show that timed regular expressions endowed with shuffle have the same expressive power as stopwatch automata. The result presented here is given for automata working over timed words. (The construction for automata over signals can be found in [Dim05].) We then restrict our

* Preliminary results were presented in [Dim05,DL07].

attention to a subclass of stopwatch automata, the partitioned stopwatch automata, in which there exist a partition of the set of stopwatches and in each location of the automaton, the set of active stopwatches is an element of this partition. This subclass, which can also be seen as a subclass of the *controlled timed automata* of [DZ98], has a decidable emptiness problem and is strictly more expressive than timed automata. We show that this subclass of stopwatch automata is language equivalent with a subclass of timed shuffle expressions, called here *fair shuffle expressions*, in which intersection can only be applied if one operand is an untimed regular expression – that is, not containing the time binding operator. We also show that partitioned stopwatch automata are not closed under intersection.

We then briefly investigate the power of diagonal constraints in partitioned stopwatch automata. Such constraints add no power when they are allowed to use only stopwatches belonging to the same class. On the contrary, without this restriction, partitioned stopwatch automata become equivalent with general stopwatch automata. We prove this result when both classes may contain constraints with rational (not integer) constants. We give an example of a partitioned stopwatch automaton with diagonal constraints using only integer constants for which we conjecture that its language is not accepted by any general stopwatch automaton which uses only integer constants.

We then introduce *distributed time-asynchronous automata*, which are tuples of timed automata synchronized on input symbols and in which time passage is local to each automaton, hence clocks in different components may be incremented with different values. Each automaton owns a set of clocks which only the owner can reset, but everyone may check the value of everyone's clocks. Discrete transitions serve for synchronization, and synchronizations take place by jointly accepting an input symbol while testing global clock constraints[1]. A component of a distributed time-asynchronous automaton may also execute internal ε-transitions without synchronization with transitions executed in other components. The distributed time-asynchronous automata are inspired from [Kri99], being an intermediary step between the distributed timed automata and the interleaved timed automata of the same cited paper [Kri99]. The class of distributed time-asynchronous automata presented here differs from the one in [DL07], in the sense that we consider only clock resets to zero.

We then show that partitioned stopwatch automata are equivalent with distributed time-asynchronous automata. The proof is done by distributing the centralized control of a partitioned stopwatch automaton in the components of a distributed time-asynchronous automaton combined with a communication mechanism between the components that ensures that each component knows exactly what transition is currently simulated and which component simulates it. The proof of this equivalence is more involved than in [DL07] where resetting a clock to 1 could be used for communication between components. The proof presented here requires a special normal form for partitioned stopwatch automata (called here *state-region determinism*), in which special restrictions apply to ε-transitions and to the relationship between the constraint and the reset of a transition and the set of stopwatches owned by its source and target states.

The paper is divided as follows: in the second section we recall the notion of stopwatch automata and we introduce our class of partitioned stopwatch automata for which the reachability problem is decidable. (This result is a corollary of results on the decidability of the emptiness

[1] Note that we do not consider here distributed alphabets, in the sense of [Zie87].

problem for Controlled Timed Automata showed in [DZ98]; we include this subsection for self-containedness concerns.) We also show that diagonal constraints added to partitioned stopwatch automata make them equivalent with general stopwatch automata. We end this section by introducing the notion of state-region deterministic partitioned stopwatch automata and prove that each partitioned stopwatch automaton can be brought to this normal form. In the third section we recall the timed shuffle expressions and prove their equivalence with stopwatch automata for timed words semantics. We then introduce the class of fair shuffle expressions and show their equivalence with partitioned stopwatch automata. The fourth section serves us for the presentation of the class of distributed time-asynchronous automata and for proving the equivalence between distributed time-asynchronous automata and partitioned stopwatch automata. We also show in this section the non closure under intersection of distributed time-asynchronous automata, which requires the introduction of the class of distributed time-asynchronous automata with private clocksets, in which each component may only read its own clocks, and not the clocks of the other components. In the fifth section we give an example of modeling, with distributed time-asynchronous automata and fair regular expressions, of Round-Robin scheduling situations and of timed processes sharing critical sections. We end with a section with conclusions.

Related Work. Timed regular expressions were first proposed in [ACM97] for regular languages of signals, and the first Kleene theorem for timed automata was proved there (see also [ACM02]). Also [BP99] and [BP01] proposed decomposition theorems for timed automata, and [Dim99] proposed regular expressions equivalent with event-clock automata. Regular expressions for stopwatch automata working on signals were first proposed in [Dim05].

Parallel compositions of timed automata were considered as early as [AD94]. The parallel composition assumes a common single time frame, and hence corresponds to intersection. [Kri99] is the first to study an interleaved composition operator on timed automata.

Decidable classes of hybrid automata have been investigated in [HKPV98,DZ98], where it has been observed that a clear partition between continuous variables having different dynamics is needed in order to have a decidable emptiness problem. Our class of partitioned stopwatch automaton can be seen as a subclass of the Controlled Timed Automata of [DZ98]. However our main concern here is not on decidability, but rather on the expressive power of this class. Some newer developments are reported in [BH09], where the reachability and model-checking problems are studied for a subclass of Controlled Timed Automata, which, when restricted to automata utilizing only diagonal constraints, is a subclass of our partitioned stopwatch automata. We also mention [ABG$^+$08], which investigates our class of distributed time-asynchronous automaton for expressiveness over untimed languages.

2 Stopwatch automata and partitioned stopwatch automata

A *timed word*, also called *timed event sequence*, is a finite sequence of nonnegative numbers and/or symbols from Σ. For example, the sequence $1.2\, a\, 1.3\, 0.4\, b$ denotes a behavior in which an *action* a occurs 1.2 time units after the beginning of the observation, and after another $1.3 + 0.4$ time units action b occurs. The length $\ell(w)$ of a timed word w is the sum of its subsequence of

reals, e.g. $\ell(1.2\,a\,1.3\,0.4\,b) = 1.2 + 1.3 + 0.4 = 2.9$. *Timed (event) languages* are then sets of timed words.

Several operations on timed words will be used in this paper. The first is concatenation, which extends the classic concatenation of untimed words by defining the concatenation of two reals as their sum. For example, $a\,1.3 \cdot 1.7\,b\,c\,0.4 = a(1.3 + 1.7)b\,c\,0.4 = a\,3\,b\,c\,0.4$. Note that both ε, the empty timed word, and the sequence consisting of the real 0 represent the *identity* of this concatenation.

The second operation on timed words is *shuffle*, which is formally defined as follows: for each two timed words w_1, w_2, $w_1 \sqcup\!\sqcup w_2 = \big\{u_1 v_1 \dots u_n v_n \mid w_1 = u_1 \dots u_n, w_2 = v_1 \dots v_n\big\}$. Both concatenation and shuffle can be straightforwardly extended to languages, so, given L_1, L_2 two timed languages: we denote $L_1 \cdot L_2 = \big\{w_1 \cdot w_2 \mid w_1 \in L_1, w_2 \in L_2\big\}$ and $L_1 \sqcup\!\sqcup L_2 = \big\{w \mid w \in w_1 \sqcup\!\sqcup w_2, w_1 \in L_1, w_2 \in L_2\big\}$.

A third useful operation on timed languages is *renaming*: it is the operation induced by a function $f : \Sigma \to \Sigma \cup \{\epsilon\}$ which syntactically replaces symbols in a timed word with other symbols, while leaving unchanged the reals composing the timed word. The renaming of $a \in \Sigma$ with $b \in \Sigma$ is denoted $[a/b]$. We also use *deletion*, which removes a symbol from a timed word, and consider it as a special case of renaming. The deletion of a symbol $a \in \Sigma$ is denoted $[a/\varepsilon]$. Hence, $[a/c, b/\varepsilon]\big(1.3\,a\,1.2\,b\,0.1\,a\big) = 1.3\,c\,1.3\,c$.

The above presentation of timed words is very convenient for constructing semantics to automata or regular expressions. However, for proofs of (in-)expressiveness, the *timed stamp* presentation is more convenient: given a timed word $w = t_1 a_1 \dots t_n a_n t_{n+1}$, its *timed stamp sequence* is the sequence of real numbers:

$$T(w) = \big(\sigma_1(w), \dots, \sigma_{n+1}(w)\big) \text{ where } \sigma_j(w) = \sum_{i=1}^{j} t_i$$

All our automata use nonnegative real-valued variables that are called *clocks* when used in timed automata, resp. *stopwatches* when used in stopwatch automata. The values of such variables may inhibit or allow taking some transition. *Clock* or *stopwatch constraints* are positive boolean combinations of elementary constraints of the type $x \in I$, with x being a (clock, resp. stopwatch) variable and $I \subseteq \mathbb{R}_{\geq 0}$ an interval with bounds in $\mathbb{N} \cup \{\infty\}$. The set of constraints with variables in a given set $\mathcal{X}$ (of *clocks* or *stopwatches*) is denoted $\mathsf{Constr}(\mathcal{X})$.

Given $v : \mathcal{X} \to \mathbb{R}_{\geq 0}$ and $C \in \mathsf{Constr}(\mathcal{X})$, we denote as usual $v \models C$ if C holds when all occurrences of each $x \in \mathcal{X}$ are replaced with $v(x)$. We also denote $v + t$ (for $t \geq 0$) the valuation $(v + t) : \mathcal{X} \to \mathbb{R}_{\geq 0}$ defined by $(v + t)(x) = v(x) + t$ for all $x \in \mathcal{X}$. *Clock*, resp. *stopwatch reset* is denoted as usual: given a valuation $v : \mathcal{X} \to \mathbb{R}_{\geq 0}$ and a subset $Y \subseteq \mathcal{X}$, we denote $v[Y := 0]$ the clock valuation defined by

$$v[Y := 0](x) = \begin{cases} 0 & \text{when } x \in Y \\ v(x) & \text{when } x \notin Y \end{cases}$$

Moreover, for $Y \subseteq \mathcal{X}$, we denote $v\big|_Y$ the valuation $v\big|_Y : Y \to \mathbb{R}_{\geq 0}$ defined by $v\big|_Y(x) = v(x)$ for all $x \in Y$. Finally, we denote $v +_Y t$ the valuation which increments all variables in Y by t,

and leaves all other variables unchanged:

$$(v +_Y t)(x) = \begin{cases} v(x) + t & \text{when } x \in Y \\ v(x) & \text{otherwise} \end{cases}$$

The set of valuations of stopwatches in $\mathcal{X}$ will be denoted in the sequel $[\mathcal{X} \to \mathbb{R}_{\geq 0}]$. Also the identically zero clock valuation in $[\mathcal{X} \to \mathbb{R}_{\geq 0}]$ is denoted $\mathbf{0}_{\mathcal{X}}$.

Definition 1. *A **stopwatch automaton** [HKPV98] is a tuple $\mathcal{A} = (Q, \mathcal{X}, \Sigma, \eta, \delta, Q_0, Q_f)$ where Q is a finite set of* locations*, $\mathcal{X}$ is a finite set of* stopwatches*, Σ is a finite set of* (action) symbols*, $Q_0, Q_f \subseteq Q$ are sets of* initial*, resp.* final *locations, $\eta : Q \to \mathcal{P}(\mathcal{X})$ gives for each state q a set of stopwatches which are* active *in q, and δ is a finite set of tuples (called* transitions*), of the form $q \xrightarrow{C,a,X} q'$, where $q, q' \in Q$, $X \subseteq \mathcal{X}$, $a \in \Sigma \cup \{\varepsilon\}$ and C is a stopwatch constraint in $\mathsf{Constr}(\mathcal{X})$.*

*A **timed automaton** is a stopwatch automaton $\mathcal{A} = (Q, \mathcal{X}, \Sigma, \eta, \delta, Q_0, Q_f)$ such that $\eta(q) = \mathcal{X}$, for any $q \in Q$.*

In the case of timed automata, the elements of $\mathcal{X}$ will be referred to as *clocks* instead of stopwatches, and the component η will be omitted in the tuple defining the automaton.

Informally, a stopwatch automaton can make time-passage transitions and discrete transitions. In a time-passage transition with duration t, the location remains unchanged while all stopwatches *that are active in that location* advance by t, and all the other stopwatches are kept unchanged. In a discrete transitions location changes. Discrete transitions are enabled when the current stopwatch valuation v satisfies the guard C of a certain transition $q \xrightarrow{C,a,X} q' \in \delta$, and when they are executed, the stopwatches in the reset component X are set to zero.

Formally, the *semantics* of a stopwatch automaton $\mathcal{A}$ is a *timed transition system* $\mathcal{T}(\mathcal{A}) = (\mathcal{Q}, \theta, \mathcal{Q}_0, \mathcal{Q}_f)$ where $\mathcal{Q} = Q \times [\mathcal{X} \to \mathbb{R}_{\geq 0}]$ represents the set of system states, $\mathcal{Q}_0 = Q_0 \times \{\mathbf{0}_{\mathcal{X}}\}$ is the set of initial states, $\mathcal{Q}_f = Q_f \times [\mathcal{X} \to \mathbb{R}_{\geq 0}]$ is the set of final states, and

$$\theta = \left\{(q, v) \xrightarrow{t} (q, v') \mid q \in Q, v' = v +_{\eta(q)} t \text{ with } t \in \mathbb{R}_{\geq 0}\right\} \tag{1}$$

$$\cup \left\{(q, v) \xrightarrow{a} (q', v') \mid \exists q \xrightarrow{C,a,X} q' \in \delta \text{ with } v \models C \text{ and } v' = v[X := 0]\right\} \tag{2}$$

Type 1 transitions are called *time-passage transitions*, whereas type 2 transitions are called *discrete transitions*. We also say that the $\mathcal{A}$-transition $\tau_{\mathcal{A}} = q \xrightarrow{C,a,X} q' \in \delta$ *generates* the $\mathcal{T}(\mathcal{A})$-transition $\tau_{\mathcal{T}(\mathcal{A})} = (q, v) \xrightarrow{a} (q', v') \in \theta$ if the two transitions are related by the conditions on the line 2 in the above definition of θ.

The **label** of a discrete transition $(q, v) \xrightarrow{a} (q', v')$ is a ($a \in \Sigma \cup \{\varepsilon\}$). The **label** of a time-passage transition $(q, v) \xrightarrow{t} (q, v')$ is $t \in \mathbb{R}_{\geq 0}$.

A **trajectory** in $\mathcal{T}(\mathcal{A})$ is a sequence $\rho = \Big((q_{i-1}, v_{i-1}) \xrightarrow{\xi_i} (q_i, v_i)\Big)_{1 \leq i \leq 2k}$ of transitions from θ, alternating between time passage and discrete, that is, $\xi_{2i-1} \in \mathbb{R}_{\geq 0}$ and $\xi_{2i} \in \Sigma \cup \{\varepsilon\}$. A **run** in $\mathcal{A}$ is a chain of transitions $\rho' = \Big(q_{i-1} \xrightarrow{C_i,a_i,X_i} q_i\Big)_{1 \leq i \leq k}$. The trajectory ρ is **associated with** the run ρ' if the length of ρ is double the length of ρ' and the $2i$-th *discrete transition* in ρ is generated by the i-th transition of ρ'.

An **accepting trajectory** in $\mathcal{T}(\mathcal{A})$ is a trajectory which starts in $\mathcal{Q}_0$, ends in $\mathcal{Q}_f$ and *does not end with a time-passage transition*. This is needed for synchronization purposes as it makes visible the end of an accepted timed word.

Remark 1. Note that any stopwatch automaton can be transformed such that for each final state $q \in Q_f$ there exists no outgoing transition leaving q. Unless stated differently, we will assume this property for all stopwatch automata considered in this paper.

An accepting trajectory **accepts** a timed word w iff w is obtained by concatenating the labels of the transitions in the trajectory. The **language accepted by** $\mathcal{A}$ is then the set of timed words which are accepted by some accepting trajectory of $\mathcal{T}(\mathcal{A})$. The language accepted by $\mathcal{A}$ is denoted $L(\mathcal{A})$. Two timed automata are **equivalent** iff they have the same language.

Definition 2. *The class of **timed shuffle languages** is the class of timed languages which are accepted by stopwatch automata.*

*The class of **timed regular languages** is the class of timed languages which are accepted by timed automata.*

The Figure 1 gives an example of a stopwatch automaton whose language is

$$L(\mathcal{A}) = \{t_1 a t_2 b t_3 c \mid t_1 + t_3 = 1\}.$$

Note that, by definition of an accepting trajectory, the automaton in Figure 1 cannot spend any time in the last state q_4 when accepting a timed word.

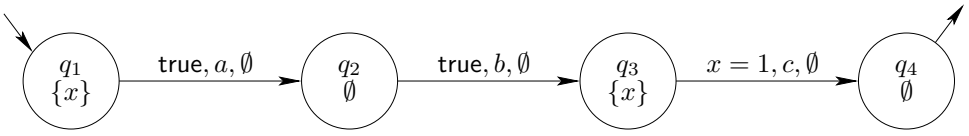

Fig. 1. A stopwatch automaton.

Remark 2. Very frequently we will abuse notation and call *trajectory* also sequences of the form $\rho = \left((q_{i-1}, v_{i-1}) \xrightarrow{\xi_i} (q_i, v_i)\right)_{1 \le i \le 2k}$ in which the alternation between time-passage transitions and discrete transitions is not satisfied. It is straightforward how to transform such a sequence into a trajectory in the sense of the above definition, either by merging two consecutive time-passage transitions in ρ, or by inserting zero time-passage transitions between two consecutive discrete transitions in ρ.

2.1 Partitioned stopwatch automata

The following subclass of stopwatch automata restricts the usage of stopwatches in a sense that makes the language emptiness problem decidable. Namely, the set of stopwatches is partitioned into classes such that, for each state q, the set of stopwatches that are active in q is a class in this partition.

Definition 3. *A **partitioned stopwatch automaton** is a stopwatch automaton in which*

$$\forall q, q' \in Q,\ \eta(q) \neq \eta(q') \Rightarrow\ \eta(q) \cap \eta(q') = \emptyset$$

An example of a stopwatch automaton is given in Figure 2, and an example of a partitioned stopwatch automaton is given in Figure 3 below.

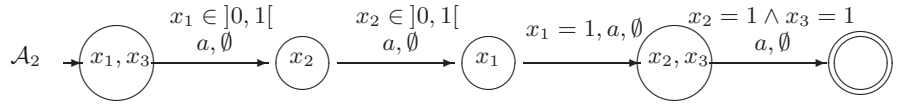

Fig. 2. A stopwatch automaton accepting $L_2 = \{t_1 a t_2 a t_3 a t_4 a \mid t_1, t_2, t_3, t_4 \in]0,1[\,, t_1 + t_3 = t_2 + t_4 = t_1 + t_4 = 1\}$

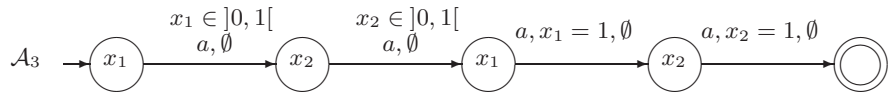

Fig. 3. A partitioned stopwatch automaton accepting $L_3 = \{t_1 a t_2 a t_3 a t_4 a \mid t_1, t_2, t_3, t_4 \in]0,1[\,, t_1 + t_3 = 1 \wedge t_2 + t_4 = 1\}$

2.2 Decidability of the reachability problem for partitioned stopwatch automata

In this section we give the generalization of the region construction from [AD94] for partitioned stopwatch automata.

We start by recalling the following result:

Theorem 1 ([HKPV98]). *The emptiness problem for stopwatch automata is undecidable.*

Turning now to decidability, recall that a *zone* (see [Yov98]) is a nonempty convex set of points in ${\mathbb{R}_{\geq 0}}^n$ which characterized by a constraint of the form $C_Z = \bigwedge_{0 \leq i,j \leq n} x_i – x_j \in I_{ij}$, where $x_0 = 0$ and I_{ij} are intervals with integer bounds. We say that C_Z is the *constraint characterizing* the zone Z. In the sequel, we consider only zones whose characterizing constraints use variables from a set of stopwatches (or clocks) $\mathcal{X}$.

An *M-region* [AD94], with $M \in \mathbb{N}$, is a zone R for which there exists a subset of variables $Y \subseteq \mathcal{X}$ such that some constraint characterizing R can be put in the following format:

$$C_R = \bigwedge\nolimits_{x \in Y} (x \in I_x) \wedge \bigwedge\nolimits_{x,y \in Y, x \neq y} \big(x - y \in I_{xy}\big) \wedge \bigwedge\nolimits_{x \in \mathcal{X} \setminus Y} \big(x \in]M, \infty[\,\big)$$

with the following properties:

- For each $x \in Y$, either $I_x = \{\alpha\}$ with $\alpha \in \mathbb{N}$, $\alpha \leq M$, or $I_x =]\alpha, \alpha+1[$ with $\alpha \in \mathbb{N}$, $\alpha \leq M-1$,
- For each $x, y \in Y$ with $x \neq y$, either $I_{xy} = \{\alpha\}$ with $\alpha \in \mathbb{Z}$, $-M \leq \alpha \leq M$, or $I_{xy} =]\alpha, \alpha+1[$ with $\alpha \in \mathbb{Z}$, $-M \leq \alpha \leq M-1$,

We denote $\mathsf{Reg}_{\mathcal{A}}(\mathcal{X})$ the set of M-regions over the stopwatches in $\mathcal{X}$ for the automaton $\mathcal{A}$, where M is the greatest constant appearing in a constraint of $\mathcal{A}$.

The following theorem adapts the well-known region construction of [AD94] for partitioned stopwatch automata:

Theorem 2. *The reachability problem for the class of partitioned stopwatch automata is decidable.*

Proof. Consider a partitioned stopwatch automaton $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \eta, \delta, Q_0, Q_f)$, denote $n = card\{\eta(q) \mid q \in Q\}$ the number of partitions of the set of stopwatches $\mathcal{X}$, and denote these partitions as $\mathcal{X}_1, \ldots, \mathcal{X}_n$. The *region automaton* corresponding with $\mathcal{A}$ is:

$$\mathcal{R}_{\mathcal{A}} = \big(Q \times \mathsf{Reg}_{\mathcal{A}}(\mathcal{X}_1) \times \ldots \times \mathsf{Reg}_{\mathcal{A}}(\mathcal{X}_n), \delta_{\mathcal{R}}, \mathcal{R}_0, \mathcal{R}_f\big)$$

where $\mathcal{R}_0 = \big\{(q_0, \mathbf{0}_{\mathcal{X}_1}, \ldots, \mathbf{0}_{\mathcal{X}_n}) \mid q_0 \in Q_0\big\}$, $\mathcal{R}_f = \big\{(q_f, R_1, \ldots, R_n) \mid q_f \in Q_f, R_i \in \mathsf{Reg}_{\mathcal{A}}(\mathcal{X}_i)\big\}$ and

$$\delta_{\mathcal{R}} = \big\{(q, R_1, \ldots, R_n) \to (q, R'_1, \ldots, R'_n) \mid \exists (q,v) \xrightarrow{\xi} (q',v') \in \theta, \xi \in \mathbb{R}_{\geq 0} \cup \Sigma \cup \{\varepsilon\}$$
$$\text{such that for all } 1 \leq i \leq n,\ R_i, R'_i \in \mathsf{Reg}_{\mathcal{A}}(\mathcal{X}_i) \text{ and } v\big|_{\mathcal{X}_i} \in R_i, v'\big|_{\mathcal{X}_i} \in R'_i\big\}$$

Here, θ is the transition relation of the timed transition system $\mathcal{T}(\mathcal{A})$ associated with $\mathcal{A}$.

It is easy to see that $L(\mathcal{A})$ is not empty if and only if $\mathcal{R}_{\mathcal{A}}$ has at least one reachable final state. □

2.3 Diagonal constraints in partitioned stopwatch automata

In this section we study the expressiveness power of diagonal constraints in partitioned stopwatch automata. Recall that diagonal constraints are elementary constraints of the type $x - y \in I$, with x and y being a (clock or stopwatch) variable and $I \subseteq \mathbb{R}_{\geq 0}$ an interval with integer bounds.

It is known that timed automata have the same expressive power with or without diagonal constraints [BDFP04]. More precisely, given a timed automaton whose constraints utilize diagonal constraints, one may algorithmically construct a timed automaton (i.e. without diagonal constraints) accepting the same language.

The same construction can be straightforwardly adapted to partitioned stopwatch automata when diagonal constraints are allowed only between stopwatches belonging to the same partition. Formally, given a partition $(\mathcal{X}_i)_{1 \leq i \leq n}$ of the set of stopwatches $\mathcal{X}$, a *$(\mathcal{X}_i)_{1 \leq i \leq n}$-compatible diagonal constraint* is a constraint of the form $x - y \in I$ for which there exists an index i with $x, y \in \mathcal{X}_i$.

Proposition 1. *The class of timed languages accepted by partitioned stopwatch automata using boolean combinations of simple constraints and $(\mathcal{X}_i)_{1 \leq i \leq n}$-compatible diagonal constraints is the same with the class of timed languages accepted by partitioned stopwatch automata without diagonal constraints.*

The proof of this proposition is an easy adaptation of results from [BDFP04].

The situation is completely different when diagonal constraints are allowed to refer to stopwatches in different components of $\mathcal{X}$. We prove here that the class of partitioned stopwatch automata with diagonal constraints and the class of stopwatch automata with diagonal constraints have the same expressive power, if *rational numbers* are allowed to be used in the constraints. As

a consequence, the emptiness problem is undecidable for partitioned stopwatch automata with (unrestricted) diagonal constraints.

Formally, define $\mathbb{Q}$-shuffle languages to be the set of timed languages which are the semantics of a shuffle regular expression in which the time-binding operator is allowed to utilize intervals with *rational* bounds also. We will show that any $\mathbb{Q}$-shuffle language can be accepted by a partitioned stopwatch automaton with diagonal constraints that may employ rational constants on its transitions.

The proof idea is presented in Figures 4 and 5 below:

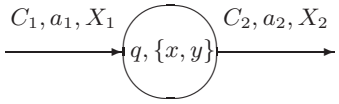

Fig. 4. A state of a stopwatch automaton with diagonal constraints

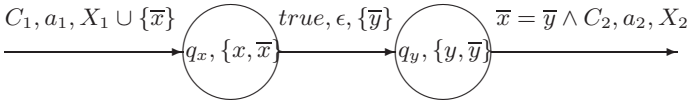

Fig. 5. Splitting the state q in two states, one which is x-active and the other y-active.

The state q in Figure 4 is split, in Figure 5 in two states, q_x and q_y, with x being active only in q_x and y active only in q_y. We append a new stopwatch $\overline{x}$ to $\eta(q_x)$ and another one $\overline{y}$ to $\eta(q_y)$, storing the time elapsed in q_x, resp. q_y. Then, when exiting q_y, it is sufficient to check that $\overline{x} = \overline{y}$ to ensure that the same interval of time has elapsed in both states q_x and q_y, which would mean that both x and y have been incremented with the same amount of time. On the other hand, the delay of staying in q_x is meant to be the same as the delay of staying in q, which means that the cumulated delay which is required for going from the transition labeled C_1, a_1, X_1 to C_2, a_2, X_2 in Figure 5 is the *double* of the delay of staying in q in Figure 4.

This replication technique can be generalized for states with more than two active stopwatches, and, if we assume that each state has a fixed number k of active stopwatches (this is possible by adding dummy stopwatches), then each state in the old automaton is split into the same number of replicas in the new automaton. The aim is to construct a new partitioned stopwatch automaton satisfying the property that the old automaton accepts a timed word w if and only if the new automaton accepts the timed word $k \otimes w$, where where $\otimes$ is the "homotety" applied to w by incrementing each time passage with a factor of k. More formally,

For $w = t_1\, a_1\, t_2 \ldots t_n\, a_n$ we define $k \otimes w$ as $k \otimes w = kt_1\, a_1\, kt_2 \ldots kt_n\, a_n$.

Therefore, to show the desired equivalence, it is sufficient to divide the constants appearing in the constraints of the initial automaton by k, to get the same language.

Theorem 3. *Partitioned stopwatch automata with diagonal constraints have the same expressive power as stopwatch automata with diagonal constraints over the class of $\mathbb{Q}$-shuffle languages.*

Proof. Take a stopwatch automaton $\mathcal{A} = (Q, \mathcal{X}, \Sigma, \eta, \delta, Q_0, Q_f)$ with diagonal constraints. Suppose, without loss of generality, that, for any $q, q' \in Q$, $card(\eta(q)) = card(\eta(q'))$. This request can be satisfied by adding dummy stopwatches. Hence, fix $k = card(\eta(q))$, for any $q \in Q$. We use the following notation: $\eta(q) = \{x_1^q, \ldots x_k^q\}$, for any $q \in Q$.

We construct a partitioned stopwatch automaton with diagonal constraints recognizing $k \otimes L(\mathcal{A})$. The partitioned stopwatch automaton with diagonal constraints has, as its set of states, pairs of the form (q, i) where $1 \leq i \leq k$ and q is a state of $\mathcal{A}$. The set $\eta(q) = \{x_1^q, \ldots x_k^q\}$ is distributed in the sequence of states $(q, 1), \ldots, (q, k)$, by letting stopwatch x_i^q be active only in the state (q, i). Furthermore, for each state (q, i) we add a new stopwatch $\overline{x}_i^q$ for storing the time elapsed in (q, i). This will be used to check that, when exiting from the last state (q, k), the same time was elapsed in each of the states (q, i) with $1 \leq i \leq k$. When exiting from the state (q, k) we enter in state $(q', 1)$ if there exists a transition from q to q' in $\mathcal{A}$.

Formally, we define $\mathcal{A}' = (Q', \mathcal{X}', \Sigma, \eta', \delta', Q'_0, Q'_f)$ as follows:

- $Q' = Q \times \{1, \ldots, k\}$, $Q'_0 = Q'_0 \times \{1\}$ and $Q'_f = Q'_f \times \{1\}$.
- $\mathcal{X}' = \mathcal{X} \cup \{\overline{x} \mid x \in \mathcal{X}\}$ and $\eta(q, i) = \{x_i^q, \overline{x}_i^q\}$;
- δ' is constructed as follows:

$$\delta'\big\{(q,k) \xrightarrow{C,a,X} (q',1) \mid \exists q \xrightarrow{C',a,X'} q' \in \delta \text{ with } C = C' \wedge \bigwedge_{i=2}^{k} x_1^q = x_i^q \text{ and } X = X' \cup \{\overline{x}_1^{q'}\}\big\}$$

$$\cup \big\{(q,i) \xrightarrow{true,\epsilon,\{\overline{x}_{i+1}^q\}} (q,i+1) \mid 1 \leq i \leq k-1\big\}$$

It is not too difficult to see that for any timed word w, $w \in L(\mathcal{A})$ if and only if $k \otimes w \in L(\mathcal{A}')$. Hence, if we further construct the automaton $\mathcal{A}'_{1/k}$ with $L(\mathcal{A}'_{1/k}) = \frac{1}{k} \otimes L(\mathcal{A}')$, then then $L(\mathcal{A}'_{1/k}) = L(\mathcal{A})$. This concludes the proof of our theorem. □

Hence we have the following corollary:

Corollary 1. *Partitioned stopwatch automata with diagonal constraints are strictly more expressive than both partitioned stopwatch automata and fair shuffle expressions.*

The reachability problem for partitioned stopwatch automata with diagonal constraints is undecidable.

As a consequence, we have the following chain of equalities and inclusions, where we denote TA the class of (timed) languages accepted by timed automata (with *rational* constraints), PSA the class of languages accepted by partitioned stopwatch automaton, $PSAD$ the class of languages accepted by partitioned stopwatch automaton with diagonal constraints, SW the class of languages accepted by stopwatch automata, and SWD the class of languages accepted by stopwatch automata with diagonal constraints:

$$TA \overset{Prop.3}{\subsetneq} PSA \overset{Prop.9}{\subsetneq} SW \overset{Conjecture}{\subsetneq} SWD \overset{Th.3\&Cor.1}{=} PSAD$$

All these identities hold for automata with rational constraints.

We conclude this section by giving an example of a stopwatch automaton with diagonal constraints for which we conjecture that there exists no stopwatch automaton without diagonal constraints which accepts the same language. The example is in Figure 6 and accepts the language $\{t_1\, a\, t_2\, a \mid t_1 = t_2\}$.

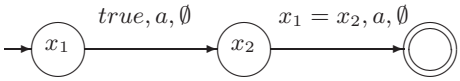

Fig. 6. A stopwatch automaton with diagonal constraints recognizing the language $\{t_1\, a\, t_2\, a \mid t_1 = t_2\}$

2.4 Some useful normal forms for partitioned stopwatch automata

Before ending this section we give a couple of normal forms for partitioned stopwatch automata. The first normal form states that each constraint and each reset on a transition concerns only stopwatches owned by the target state. The second normal form is defined by the following properties:

- No ε-transition can be preceded or succeded in zero time by another discrete transition.
- All sequences of transitions with symbols in Σ that are taken in zero time must be taken in the same component, which is the component containing the first state in the sequence.
- Discrete transitions are *deterministic* in the following sense: no two discrete transitions may lead to two distinct states in $\mathcal{T}(\mathcal{A})$, both owning the same class of stopwatches and such that the stopwatch values right after the transition lie in the same regions.

We also need the following notion: given a stopwatch automaton $\mathcal{A}$ and a stopwatch x of $\mathcal{A}$, a location q in $\mathcal{A}$ is called *x-active* if $x \in \eta(q)$; otherwise, we will say q is *x-inactive*. A run whose intermediate locations are x-inactive locations is called an *x-inactive run*.

We begin with the statement and proof of the first normal form:

Lemma 1. *Given a partitioned stopwatch automaton $\mathcal{B} = (Q, \mathcal{X}, \Sigma, \eta, \delta, Q_0, Q_f)$ there exists a partitioned stopwatch automaton $\mathcal{B}' = (Q', \mathcal{X}, \Sigma, \eta', \delta', Q'_0, Q'_f)$ which has the same language as $\mathcal{B}$ and has the following properties:*

- *For each transition $q \xrightarrow{C,a,X} r \in \delta'$ with $r \notin Q'_f$, we have that $X \subseteq \eta'(r)$ and C does not constrain the clocks which are inactive in q – that is, $C \wedge (x = \alpha)$ is satisfiable for each $x \notin \eta'(q)$ and $\alpha \in \mathbb{R}_{\geq 0}$.*
- *$\mathcal{B}'$ satisfies the condition in Remark 1, i.e. no outgoing transitions are leaving final states.*
- *$\eta'(q) = \emptyset$ for each $q \in Q_f$.*

Proof. The technique used to construct the automaton $\mathcal{B}'$ is to remove all the constraints of the form $(x \in I)$ from the label of transitions $q \xrightarrow{C,a,X} r$ for which x is inactive in q, and shift them to the transitions that leave r. Since the value of x does not change along an x-inactive run, the conjunction of all the constraints on x along such a run only needs to be checked at the end of the run. Some particular care needs to be taken when the automaton has circuits in which x is

inactive. But, fortunately, we only need to remember whether a run passes at least once through a transition, further passages through the same transition remain irrelevant for the above idea of x-constraint shifting.

Resets are also shifted similarly: each reset of a stopwatch x at the beginning of a run which passes through x-inactive states can be shifted from the first transition of this run to its last transition.

Another special care has to be taken with final states: constraints labeling transitions whose target states are final should not be removed. But, by assuming that all final states have no outgoing transitions, we may gather all these states in a single component in which no stopwatch is active, such that the first property be satisfied also for transitions entering final states.

In the sequel we construct the automaton $\mathcal{B}'$ which satisfies the required properties for a single stopwatch x. The whole construction can be iterated for the rest of stopwatches. We use the following notations:

1. We denote $I_{\mathcal{B}}$ the set of intervals for which $x \in I$ occur on some transition in $\mathcal{B}$.
2. Given a constraint C and a clock x, we denote $C \setminus x$ the constraint which is obtained from C by removing the atomic constraints referring to x.

Formally, $\mathcal{B}' = (Q' \cup \overline{Q}_f, \mathcal{X}, \Sigma, \eta', \delta', Q'_0, \overline{Q}_f)$ with:

– $Q' = Q \times I_{\mathcal{B}} \times \{0,1\}$ and $\overline{Q}_f$ is a copy of Q_f, $\overline{Q}_f = \{\overline{r} \mid r \in Q_f\}$.
In a $\mathcal{B}'$-state (q, θ, i), the third component records the passage through a transition resetting x during the x-inactive run which ends in q, that is, $i = 1$ if there exists such a resetting transition and $i = 0$ otherwise. When $i = 0$, the second component θ records the intervals which constrained the clock x along the x-inactive run that ends in q. For $i = 1$, the same component records the intervals which constrained x before the x-reset. (The constraints that occur *after* an x-reset are not recorded, since the only relevant constraint on x after this reset is $x = 0$.)
– $\eta'(q, \theta, i) = \eta(q)$ for $(q, \theta, i) \in Q'$ and $\eta'(\overline{q}) = \emptyset$ for $\overline{q} \in \overline{Q}_f$.
– $Q'_0 = Q_0 \times \{\emptyset\} \times \{0\}$.
– The transition relation is composed of two types of transitions:
  - $(q, \theta_1, i_1) \xrightarrow{C,\xi,X} (r, \theta_2, i_2)$, if there exists $q \xrightarrow{C',\xi,X'} r \in \delta$ such that

$$i_2 = \begin{cases} 0 & \text{if } r \text{ is } x\text{-active} \\ 1 & \text{if } r \text{ is } x\text{-inactive and } x \in X' \\ i_1 & \text{otherwise} \end{cases}$$

$$C = \begin{cases} C' \setminus x & \text{if } q \text{ is } x\text{-inactive} \\ C' \wedge \bigwedge_{I \in \theta_1} (x \in I) & \text{otherwise} \end{cases}$$

$$\theta_2 = \begin{cases} \emptyset & \text{if } q \text{ is } x\text{-active} \\ \theta_1 \cup \{x \in I \mid x \text{ occurs in } C' \text{ and } i_1 = 0\} & \text{otherwise} \end{cases}$$

  - $(q, \theta, i) \xrightarrow{C,\xi,X} \overline{r}$ for $r \in Q_f$ and there exists $\tau = q \xrightarrow{C',\xi,X'} r \in \delta$ such that $C = C' \wedge \bigwedge_{I \in \theta} (x \in I)$ and $X = X' \cup \{x\}$ if $i = 1$, $X = X'$ otherwise.

The correctness of the construction relies on the fact that the occurrence of the same transition along an x-inactive path does not need to be memorized more than once when computing the second component of each state $(q, \theta, i) \in Q'$. □

The second normal form is formalized in the following definition:

Definition 4. *A **state-region deterministic partitioned stopwatch automaton** is a tuple $\mathcal{A} = (Q, \mathcal{X}, \Sigma, \eta, \delta, Q_0, Q_f)$, whose associated timed transition system $\mathcal{T}(\mathcal{A}) = (\mathcal{Q}, \theta, \mathcal{Q}_0, \mathcal{Q}_f)$ satisfies the following properties:*

1. Final states have no outgoing transitions*, i.e., $\mathcal{A}$ satisfies the property in Remark 1.*
2. *Two transitions starting from the same state and labeled with the same region constraint must lead to states associated with distinct classes of stopwatches – we call this property* location determinism.
   More formally, for any states $(q, v), (q_1, v'), (q_2, v') \in \mathcal{Q}$ and each $\xi \in \Sigma \cup \{\varepsilon\}$, if $\eta(q_1) = \eta(q_2)$, $(q, v) \xrightarrow{\xi} (q_1, v')$ and $(q, v) \xrightarrow{\xi} (q_2, v')$ then $q_1 = q_2$.
3. *There exists a unique initial state $q_0 \in Q_0$ such that (a) all timed words that begin with a sequence of discrete transitions separated by intervals of length zero must be parsed starting from q_0, (b) all timed words that start with a non-zero time passage must be parsed starting from initial states different from q_0, and (c) for each class of stopwatches $\mathcal{X}_i$ there exists at most one initial state which is labeled with $\mathcal{X}_i$. We call this property* initial trajectory determinism
   More formally :
   (a) For each trajectory $(q_0, \mathbf{0}_\mathcal{X}) \xrightarrow{0} (q, \mathbf{0}_\mathcal{X}) \xrightarrow{\xi} (q', \mathbf{0}_\mathcal{X})$ with $q_0 \in Q_0$ we must have $q = q_0$.
   (b) For each trajectory $(q, \mathbf{0}_\mathcal{X}) \xrightarrow{t} (q, v) \xrightarrow{\xi} (q', v')$ with $t > 0$ and $q \in Q_0$ we must have $q \neq q_0$.
   (c) For each pair of trajectories $(q_1, \mathbf{0}_\mathcal{X}) \xrightarrow{t} (q_1, v) \xrightarrow{\xi} (q', v')$ and $(q_2, \mathbf{0}_\mathcal{X}) \xrightarrow{t} (q_2, v) \xrightarrow{\xi} (q', v')$, if $\eta(q_1) = \eta(q_2)$ and $q_1, q_2 \in Q_0$ then $q_1 = q_2$.
4. *If two consecutive discrete transitions are taken in zero time, then none of them may be labeled with ε, that is, for each trajectory in $\mathcal{T}(\mathcal{A})$, $(q_1, v_1) \xrightarrow{\xi_1} (q_2, v_2) \xrightarrow{\xi_2} (q_3, v_3)$ with $\xi_1, \xi_2 \notin \mathbb{R}_{\geq 0}$, we have that $\xi_1, \xi_2 \in \Sigma$.*
5. *In any sequence of discrete transitions separated by zero time, the source state of each transition owns the same set of stopwatches. Formally, for each trajectory in $\mathcal{T}(\mathcal{A})$, $(q_1, v_1) \xrightarrow{a_1} (q_2, v_2) \xrightarrow{a_2} (q_3, v_3)$ with $a_1, a_2 \in \Sigma$, we have that $\eta(q_1) = \eta(q_2)$.*

The second normal form is ensured by the following:

Lemma 2. *Given a partitioned stopwatch automaton $\mathcal{A}$ we can effectively compute a state-region deterministic partitioned stopwatch automaton $\mathcal{B}$ with $L(\mathcal{A}) = L(\mathcal{B})$.*

Proof. We will start with a partitioned stopwatch automaton and modify it in four stages, each stage adding one extra property between the five above. First, by means of Lemma 1, we assume that each transition $q \xrightarrow{C,\xi,X} r$ has the property that C and X refer only to the stopwatches that are active in r.

Secondly, we assume that $\mathcal{A}$ is in fact transformed using the region construction for partitioned stopwatch automata. More formally, we assume that, for any transition $q \xrightarrow{C,\xi,X} r \in \delta$ with $\eta(r) = \mathcal{X}_i$, there exists a region $R_i \in \mathsf{Reg}_{\mathcal{A}}(\mathcal{X}_i)$ such that C is a defining constraint for R_i, and also each transition in $\mathcal{A}$ is *reachable*, that is, there exists an accepting run that contains that transition and which is associated with at least one accepting trajectory.

Thirdly, we will assume that each component $\mathcal{X}_i$ contains a distinguished stopwatch x_i^0 that is reset before entering each state $q \in Q$ owning $\mathcal{X}_i$ (i.e. $\eta(q) = \mathcal{X}_i$). Hence, on each transition $q \xrightarrow{C,\xi,X} r \in \delta$, the (region) constraint C either implies $x_i^0 = 0$ or is such that $C \wedge (x_i^0 = 0)$ is unsatisfiable. For convenience, for any set of stopwatches X, we denote $C\big|_X$ the subconstraint of C involving only clocks in X; also $C \subseteq (x \in I)$ denotes situations in which the constraint $C \wedge (x \notin I)$ is unsatisfiable.

The fourth property in the definition of state-region determinism can be obtained as follows: we remove all transitions $q \xrightarrow{C,\varepsilon,X} r$ with $C\big|_{x_i^0} = (x_i^0 = 0)$ where $\eta(r) = \mathcal{X}_i$ (which are exactly the ε-transitions taken after a time lapse of 0 in source state q), by replacing them with other constraints that simulate sequences of ε-transitions that are taken in zero time. Formally, the construction is the following: denote first

$$\delta_\varepsilon = \{q \xrightarrow{C,\varepsilon,X} r \mid \exists i \text{ such that } \eta(r) = \mathcal{X}_i \text{ and } C\big|_{x_i^0} = (x_i^0 = 0)\}$$

Then we replace δ with the set of transitions δ'' defined as follows:

- $q \xrightarrow{C,\xi,X} r \in \delta''$ if either $q \xrightarrow{C,\xi,X} r \in \delta \setminus \delta_\varepsilon$ or there exists a run $(q_{j-1} \xrightarrow{C_j,\xi_j,X_j} q_j)_{1\le j\le m}$ in $\mathcal{A}$ with $\xi_{j_0} \in \Sigma$ for some $j_0 \le m$ and $\xi_j = \varepsilon$ for all $j \ne j_0$, $\xi_m = \xi$, $q_0 = q$, $q_m = r$, $C = \bigwedge_{1\le j\le m} C_j$ and if we denote i_j the index with $\eta(q_j) = \mathcal{X}_{i_j}$ then $C_1\big|_{x_{i_1}^0} \subseteq (x_{i_1}^0 > 0)$ and for all $j \ge 2$, $C_j\big|_{x_{i_j}^0} = (x_{i_j}^0 = 0)$.

Also we replace Q_0 with a new set of initial states Q_0' defined as follows:

$$Q_0' = \big\{r \in Q \mid \text{ there exists a run } \big(q_{j-1} \xrightarrow{C_j,\xi_j,X_j} q_j\big)_{1\le j\le m} \text{ in } \mathcal{A} \text{ with } q_0 \in Q_0, q_m = r, \text{ and for all } j, \xi_j = \varepsilon, C_j \models 0_{\mathcal{X}}\big\}$$

Note that the automaton obtained still satisfies the property in Remark 1.

The fifth property in Definition 4 is obtained by modifying the automaton obtained above along the following ideas: first, whenever a transition $q \xrightarrow{C,a,X} r$ with some constraint $x_i^0 = 0$ is taken (which, by the above construction, means that $a \ne \varepsilon$), its target state will be redirected to a copy of r which will be associated with the same set of stopwatches as q. In order to keep the properties from Lemma 1 satisfied after this, we need to record also in this state the constraint C and the reset component X, so as to postpone their satisfaction until the moment when the set of stopwatches $\eta(r)$ will be encountered again, or until a final state is reached.

Formally, considering that we start with an automaton $\mathcal{A}$ satisfying property 4 in Definition 4 and the properties in Lemma 1, we build the partitioned stopwatch automaton $\tilde{\mathcal{A}} = (\tilde{Q}, \Sigma, \tilde{\mathcal{X}}, \tilde{\delta}, \tilde{\eta}, \tilde{Q}_0, \tilde{Q}_f)$ with

1. $\tilde{Q} = \big\{(q, C, X, i) \mid q \in Q, C$ occurs on some transition of $\delta, X \subseteq \mathcal{X}, 0 \leq i \leq n\big\}$.
   States of the type $(q, C, X, 0)$ are **transient** (that is, control may stay only for 0 time units in such states), and occur during sequences of transitions taken in zero time, starting from component i. States of the type (q, C, X, i) with $1 \leq i \leq n$ are **persistent** and have the property that all transitions leaving such states must ensure that $x_i^0 > 0$ where $\eta(q) = \mathcal{X}_i$.
2. $\tilde{Q}_0 = \{(q, C, X, i) \in \tilde{Q} \mid q \in Q_0, 0 \leq i \leq n\}$.
3. $\tilde{Q}_f = \{(q, C, X, i) \in \tilde{Q} \mid q \in Q_f, 0 \leq i \leq n\}$.
4. $\tilde{\eta}(q, C, X, i) = \mathcal{X}_i$ for $1 \leq i \leq n$, and $\tilde{\eta}(q, C, X, 0) = \eta(q)$.
5. The transition relation is composed of tuples of the following types:
   (a) $(q, C_1, X_1, i_1) \xrightarrow{C,a,X} (r, C_2, X_2, i_2)$ if $i_1 \geq 1$ and there exists $q \xrightarrow{C',a,X'} r \in \delta$ such that $C = C'\big|_{\mathcal{X}_{i_1}} \wedge (x_{i_1}^0 = 0)$, $C_2 = C_1 \wedge C'\big|_{\mathcal{X} \setminus \mathcal{X}_{i_1}}$,

$$i_2 = \begin{cases} i_1, & \text{if } \eta(q) \neq \mathcal{X}_{i_1} \\ \text{random choice between } 0 \text{ and } i_1, & \text{otherwise} \end{cases}$$

$$X = \begin{cases} (X' \cap \mathcal{X}_{i_1}) \cup \{x_{i_1}^0\}, & \text{if } i_1 = i_2 \\ (X' \cap \eta(r)) \cup \{x_j^0\}, & \text{if } i_2 = 0 \text{ and } \eta(r) = \mathcal{X}_j \end{cases}$$

$$X_2 = \begin{cases} X_1 \cup (X \setminus \mathcal{X}_{i_1}) & \text{if } i_1 = i_2 \\ X_1 \cup (X \setminus \eta(r)) & \text{otherwise} \end{cases}$$

   (b) $(q, C_1, X_1, 0) \xrightarrow{C,a,X} (r, C_2, X_2, i)$ if there exists $q \xrightarrow{C',a,X'} r \in \delta$ such that if we denote j_1, j_2 the indices with $\eta(q) = \mathcal{X}_{j_1}$ and $\eta(r) = \mathcal{X}_{j_2}$ then $i \in \{0, j_1, j_2\}$ is an arbitrary value, $C = C'\big|_{\eta(q)} \wedge C_1\big|_{\eta(q)} \wedge (x_{j_1}^0 > 0)$, $C_2 = C_1\big|_{\mathcal{X} \setminus \mathcal{X}_i} \wedge C'\big|_{\mathcal{X} \setminus \mathcal{X}_{j_1}}$,

$$X = \begin{cases} (X_1 \cup X' \cup \{x_i^0\}) \cap \mathcal{X}_i & \text{if } i \neq 0 \\ (X_1 \cup X' \cup \{x_{j_2}^0\}) \cap \eta(r) & \text{otherwise} \end{cases}$$

$$X_2 = \begin{cases} (X_1 \cup X) \setminus \mathcal{X}_i & \text{if } i \neq 0 \\ (X_1 \cup X) \setminus \eta(r) & \text{otherwise} \end{cases}$$

Remark 3. Note that the above construction has the property that the set of initial states is partitioned, $\tilde{Q}_0 = Q_0^t \cup Q_0^p$ with $Q_0^t \cap Q_0^p = \emptyset$ and such that for each $q \in \tilde{Q}_0$, if $(q, \mathbf{0}_{\mathcal{X}}) \xrightarrow{t} (q, v) \xrightarrow{\xi} (q', v')$ is a trajectory in $\mathcal{T}(\tilde{\mathcal{A}})$ with $q \in \tilde{Q}_0$ then $t = 0$ if and only if $q \in \tilde{Q}_0^t$.

Also, the property in Remark 1 is preserved by this construction, with the extra property that there exists a unique stopwatch that is active only in the final states.

Finally, the properties 2 and 3 in Definition 4 can be obtained by a subset construction: starting with the automaton $\tilde{\mathcal{A}}$ already satisfying properties 4 and 5, we build the automaton $\hat{\mathcal{A}} = (\hat{Q}, \Sigma, \hat{\mathcal{X}}, \hat{\delta}, \hat{\eta}, \hat{Q}_0, \hat{Q}_f)$ with

1. $\hat{Q} = \{S \subseteq \tilde{Q} \mid$ there exists i with $\eta(q) = \mathcal{X}_i$ for all $q \in S\} \cup \{\overline{S} \mid S \subseteq \tilde{Q}\}$.
   The last type of macro-states is used at the beginning of trajectories which start with a sequence of discrete transitions taken in zero time, hence the set of stopwatches which is associated with these macro-states is not essential.

2. $\hat{Q}_0 = \{\overline{Q_0^t}\} \cup \{S \subseteq Q_0^p \mid \exists i \text{ with } \eta(q) = \mathcal{X}_i \text{ for all } q \in S\}$.
3. $\hat{Q}_f = \{S \in \hat{Q} \mid S \cap Q_f \neq \emptyset\}$.
4. $\hat{\mathcal{X}} = \mathcal{X} \cup \{\overline{x}\}$, where $\overline{x}$ is an extra stopwatch used only in the macro-states at the beginning of trajectories which start with a sequence of discrete transitions taken in zero time.
   We also partition $\hat{\mathcal{X}}$ as $\hat{\mathcal{X}} = \bigcup_{1 \leq i \leq n+1} \hat{\mathcal{X}}_i$ with $\hat{\mathcal{X}}_i = \mathcal{X}_i$ for $i \leq n$ and $\hat{\mathcal{X}}_{n+1} = \{\overline{x}\}$.
5. For each $S \in \hat{Q}$, $\eta(S) = \eta(q)$ for some $q \in S$, and for each $\overline{S} \in \hat{Q}$, $\eta(\overline{S}) = \mathcal{X}_{n+1} = \{\overline{x}\}$ (note the comment on the first item above).
6. The transition relation is:

$$\hat{\delta} = \left\{S \xrightarrow{C,a,X} R \mid \exists i, 1 \leq i \leq n \text{ s.t. } R = \{r \in \tilde{Q} \mid \eta(r) = \mathcal{X}_i, \exists q \in S \text{ with } q \xrightarrow{C,a,X} r\}\right\}$$
$$\cup \left\{\overline{S} \xrightarrow{(\overline{x}=0),a,X} R \mid R = \{r \in Q \mid \text{ there exists } q \in S \text{ with } q \xrightarrow{C,a,X} r \right.$$
$$\left. \text{and } C \wedge (x = 0) \text{ is satisfiable for each } x \in \mathcal{X}\}\right\}$$

Note again that the resulting automaton satisfies the condition in Remark 1, since, due to Remark 3 above, final states can only be grouped with other final states in $\hat{Q}$. This ends the proof of Lemma 2. □

3 Regular expressions for stopwatches

In this section we present the class of timed shuffle expressions and their equivalence with stopwatch automata. We also give a class of timed shuffle expressions which are equivalent with partitioned stopwatch automata.

Definition 5. *The class of **timed shuffle expressions** is the following:*

$$E ::= a \mid \underline{\mathbf{t}} \mid E + E \mid E \cdot E \mid E \wedge E \mid E^* \mid \langle E \rangle_I \mid E \sqcup\!\sqcup E \mid [a/b]E$$

where , $a \in \Sigma$, $b \in \Sigma \cup \{\varepsilon\}$ and I is an interval with a nonnegative integer bound and a nonnegative integer or infinite upper bound.

*A **timed regular expression** is a timed shuffle expression constructed without the operator $\sqcup\!\sqcup$. An **unconstrained expression** is a timed shuffle expression constructed without the operator $\langle \_ \rangle_I$.*

The $\langle \_ \rangle_I$ operator reads as the *time binding operator*.

The *semantics* of a timed shuffle expression is given by the following rules:

$$\|a\| = \{a\} \qquad \|\underline{\mathbf{t}}\| = \{t \mid t \in \mathbb{R}_{\geq 0}\}$$
$$\|E_1 + E_2\| = \|E_1\| \cup \|E_2\| \qquad \|E^*\| = \|E\|^*$$
$$\|E_1 \wedge E_2\| = \|E_1\| \cap \|E_2\| \qquad \|\langle E \rangle_I\| = \{w \in \|E\| \mid \ell(w) \in I\}$$
$$\|E_1 \cdot E_2\| = \|E_1\| \cdot \|E_2\| \qquad \|E_1 \sqcup\!\sqcup E_2\| = \|E_1\| \sqcup\!\sqcup \|E_2\|$$
$$\|[a/b]E\| = \{[a/b](w) \mid w \in \|E\|\}$$

where $a \in \Sigma$ and $b \in \Sigma \cup \{\varepsilon\}$.

Example 1. The expression $E_1 = [z_1/\varepsilon][z_2/\varepsilon]\big((\langle z_1 \underline{\mathbf{t}} a z_1 \underline{\mathbf{t}} c\rangle_1 \sqcup\!\sqcup z_2 \underline{\mathbf{t}} b) \wedge z_1 \underline{\mathbf{t}} a z_2 \underline{\mathbf{t}} b z_1 \underline{\mathbf{t}} c\big)$ represents the language of the automaton $\mathcal{A}$ from Figure 1.

Note, in this example, the need to "duplicate" each symbol: if we did not use the additional symbols z_1 and z_2, we would not be able to correctly "insert" the first subexpression of the shuffle "within" the second, i.e., the shuffle expression $E_2 = \big(\langle \underline{\mathbf{t}} a \underline{\mathbf{t}} c\rangle_1 \sqcup\!\sqcup \underline{\mathbf{t}} b\big) \wedge \underline{\mathbf{t}} a \underline{\mathbf{t}} b \underline{\mathbf{t}} c$, is not equivalent with the automaton in Figure 1. To see this, note that the shuffle expression E_2 is equivalent with the timed regular expression $E_3 = \langle \underline{\mathbf{t}} a \underline{\mathbf{t}} b \langle \underline{\mathbf{t}}\rangle_{[0,1]} c\rangle_{[1,\infty[}$, which is obviously not equivalent with the stopwatch automaton in Figure 1. The problem lies in the fact that the duration before b in the shuffled subexpression $\underline{\mathbf{t}} b$ must not "mix" with the other durations. The use of the additional symbols z_1 and z_2 in Example 1 is essential in forbidding this mixing.

Remark 4. Note that an unconstrained expression still has some timing information, because it may contain two adjacent symbols from Σ, which would mean that the two symbols have to be separated by a zero time delay. For example, all the timed words that are in the semantics of the expression $\underline{\mathbf{t}} a b \underline{\mathbf{t}} c$ have the property that a and b occur "at the same time", with a preceding b, in the sense of *weakly monotonic time* of [PH98].

We recall first the following theorem from [ACM02]:

Theorem 4. *Timed regular expressions have the same expressive power as timed automata.*

We will use also the following result which relates unconstrained expressions with a special type of timed automata, called here **zero-constrained timed automata**. These are timed automata containing a single clock which is reset on every transition and whose constraints are only true or $x = 0$.

Proposition 2 (Kleene theorem for zero-constrained timed automata). *The class of unconstrained expressions is equivalent with the class of zero-constrained timed automata.*

Proof. Let us first observe that zero-constrained timed automata are a strict subclass of the real-time automata from [Dim01], which are timed automata with a single clock which is reset on each transition. This means that one may actually construct a real-time regular expression in the sense of [Dim01], which will only contain time binding operators of the type $\langle\cdot\rangle_0$ or $\langle\cdot\rangle_{[0,\infty[}$.

The last type of time binding operators can be removed from the expressions. On the other hand, note that, for any timed regular expression E, if we denote $u(E)$ the expression resulting by removing all $\underline{\mathbf{t}}$ atoms from E, then:

$$\|\langle E\rangle_0\| = \|u(E)\|$$

This means that the operators $\langle\cdot\rangle_0$ are also expressible without time binding operators, fact which ends the proof of the inverse inclusion.

For the direct inclusion, it's not hard to see then that for each timed regular expression not involving the time-passage symbol $\underline{\mathbf{t}}$, we can construct a zero-constrained timed automaton which tests, at each transition, that the clock is zero, and resets it immediately.

On the other hand, for the expression $\underline{\mathtt{t}}$ the construction of an equivalent timed automaton actually builds a zero-constrained timed automaton which has a true constraint when entering its final states.

Then, the constructions for union, concatenation and star for real-time automata in [Dim01] also apply for zero-constrained timed automata, and the results are still zero-constrained timed automata. The construction for intersection is also easily adaptable from the case of classical finite automata, and still yields zero-constrained timed automata. It only remains to give a construction proving that shuffle of zero-constrained timed automata yields zero-constrained timed automata too.

To this end, consider two zero-constrained timed automata $\mathcal{A}_1 = \big(Q_1, \Sigma, \{x_1\}, \delta_1, Q_0^1, Q_f^1\big)$ and $\mathcal{A}_2 = \big(Q_2, \Sigma, \{x_2\}, \delta_2, Q_0^2, Q_f^2\big)$. By state-splitting, we may ensure that each state $q_1 \in Q_1$ is either transient, hence all its outgoing transitions are labeled with $(x_1 = 0)$, or all its outgoing transitions are labeled with true. The same can be ensured for $\mathcal{A}_2$.

The automaton recognizing $L(\mathcal{A}_1) \text{ш} L(\mathcal{A}_2)$ is $\mathcal{A} = \big(Q_1 \times Q_2, \Sigma, \{x\}, \delta, Q_0, Q_f\big)$ where:

– $Q_0 = Q_0^1 \times Q_0^2$ and $Q_f = Q_f^1 \times Q_f^2$.
– The transition relation is composed of the following tuples:
  - $(q_1, q_2) \xrightarrow{x \in I, a, \{x\}} (q_1', q_2)$ if $q_1 \xrightarrow{x_1 \in I, a, \{x_1\}} q_1' \in \delta_1$ and q_2 is persistent.
  - $(q_1, q_2) \xrightarrow{x \in I, a, \{x\}} (q_1, q_2')$ if $q_2 \xrightarrow{x_2 \in I, a, \{x_2\}} q_2' \in \delta_2$ and q_1 is persistent.
  - $(q_1, q_2) \xrightarrow{(x=0), a, \{x\}} (q_1', q_2')$ if both q_1 and q_2 are transient.

  Note that a tuple (q_1, q_2) formed of a transient state in $\mathcal{A}_1$ and a persistent state in $\mathcal{A}_2$ becomes a persistent state in $\mathcal{A}$.

This ends the proof of this proposition. □

The following theorem gives the generalization of Asarin, Caspi & Maler's result [ACM02] for the class of timed shuffle languages.

Theorem 5. *Timed shuffle expressions have the same expressive power as stopwatch automata.*

Proof. For the direct inclusion, the union, intersection, concatenation, star, time binding and renaming constructions from [ACM97,ACM02] can be easily extended to stopwatch automata. We will only give here the construction for the shuffle of two stopwatch automata, which is a straightforward generalization of the construction at the end of Proposition 2.

So take two automata $\mathcal{A}_i = (Q_i, \mathcal{X}_i, \Sigma, \eta_i, \delta_i, Q_0^i, Q_f^i)$ $(i = 1, 2)$ with $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$. The automaton accepting $L(\mathcal{A}_1) \text{ш} L(\mathcal{A}_2)$ is then $\mathcal{A} = (Q, \mathcal{X}, \Sigma, \eta, \delta, Q_0, Q_f)$ where

– $Q = Q_1 \times Q_2 \times \{1, 2\}$ and $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2$.
– $Q_0 = Q_0^1 \times Q_0^2 \times \{1, 2\}$ and $Q_f = Q_f^1 \times Q_f^2 \times \{1, 2\}$.
– $\eta : Q \to \mathcal{P}(\mathcal{X}_1 \cup \mathcal{X}_2)$ is defined by $\eta(q_1, q_2, i) = \eta_i(q_i)$.
– The transition relation is defined as follows:

$$\begin{aligned}\delta = &\big\{(q_1, q_2, 1) \xrightarrow{C, a, X} (q_1', q_2, 1) \mid q_1 \xrightarrow{C, a, X} q_1' \in \delta_1, a \in \Sigma \cup \{\varepsilon\}\big\} \\ &\cup \big\{(q_1, q_2, 2) \xrightarrow{C, a, X} (q_1, q_2', 2) \mid q_2 \xrightarrow{C, a, X} q_2' \in \delta_2, a \in \Sigma \cup \{\varepsilon\}\big\} \\ &\cup \big\{(q_1, q_2, 1) \xrightarrow{\mathsf{true}, \varepsilon, \emptyset} (q_1, q_2, 2), (q_1, q_2, 2) \xrightarrow{\mathsf{true}, \varepsilon, \emptyset} (q_1, q_2, 1) \mid q_1 \in Q_1, q_2 \in Q_2\big\}\end{aligned}$$

The proof of the reverse inclusion is a two-step proof: the first step involves the decomposition of each stopwatch automaton with n stopwatches into an intersection of n one-stopwatch automata – similarly with the proof of the Kleene theorem for timed automata [ACM02]. The second step shows how to construct a timed shuffle expression equivalent with an automaton with one stopwatch.

The decomposition step requires a preliminary relabeling of the transitions of $\mathcal{A}$ such that two different transitions bear different labels. This relabeling is done in the classical way [ACM97]: we use δ as the new set of transition labels, hence obtaining the stopwatch automaton $\tilde{\mathcal{A}} = (Q, \mathcal{X}, \delta, \eta, \tilde{\delta}, Q_0, Q_f)$ in which

$$\tilde{\delta} = \left\{ q \xrightarrow{C,q \xrightarrow{C,a,X} r,X} r \mid q \xrightarrow{C,a,X} r \in \delta \right\} \tag{3}$$

Then $L(\mathcal{A}) = \lambda(L(\tilde{\mathcal{A}}))$ where λ is the renaming defined by $\lambda : \delta \to \Sigma \cup \{\varepsilon\}, \lambda(q \xrightarrow{C,a,X} r) = a$.

We may then decompose $\tilde{\mathcal{A}}$ into n automata $\mathcal{A}_i = (Q, \{x_i\}, Q, \eta_i, \delta_i, Q_0, Q_f)$ having a single stopwatch, with $\eta_i(q) = \eta(q) \cap \{x_i\}$ and δ_i is a copy of $\tilde{\delta}$ in which the guard and the reset component of each tuple is a projection on $\{x_i\}$. Hence

$$L(\mathcal{A}) = \lambda\big(L(\mathcal{A}_1) \cap \ldots \cap L(\mathcal{A}_n)\big)$$

The proof of this identity is similar to the case of timed automata [ACM02].

For the second step, suppose that $\mathcal{B} = (Q, \{x\}, \Sigma, \eta, \delta, Q_0, Q_f)$ is an automaton with a single stopwatch in which all transitions have distinct labels.

Let us note first the following corollary of Lemma 1:

Corollary 2. *Given a one-stopwatch automaton $\mathcal{B} = (Q, \{x\}, \Sigma, \eta, \delta, Q_0, Q_f)$ there exists a one-stopwatch automaton $\mathcal{B}' = (Q', \{x\}, \Sigma, \eta', \delta', Q'_0, Q'_f)$ which has the same language as $\mathcal{B}$ and in which for each pair of locations $q, r \in Q'$ with x being inactive in both q and r, if $q \xrightarrow{C,a,X} r \in \delta'$ and $r \notin Q_f$ then $C = \mathsf{true}$ and $X = \emptyset$.*

This result follows easily if we observe that a one-stopwatch automaton is in fact a partitioned stopwatch automaton. We will use this result in our proof of the reverse inclusion as follows:

We wish to decompose $\mathcal{B}$ into three automata such that:

$$L(\mathcal{B}) = \big(L(\mathcal{B}_1) \sqcup\!\sqcup L(\mathcal{B}_2)\big) \cap L(\mathcal{B}_3) \tag{4}$$

In this decomposition, $\mathcal{B}_2$ is an untimed automaton, $\mathcal{B}_3$ is a zero-constrained timed automaton, and $\mathcal{B}_1$ is a "one-and-a-half-clock" timed automaton, that is, a timed automaton with two clocks in which one of the clocks can only be compared with zero, and, in this sense, serves only for checking that some states are transient. $\mathcal{B}_1$ will carry the duration constraints of x (the stopwatch of $\mathcal{B}$), while $\mathcal{B}_2$ will carry the sequential properties within the states in which x is inactive.

The task of $\mathcal{B}_3$ is to correctly connect the sequences of states in which x is active with those in which x is inactive. $\mathcal{B}_3$ removes, similarly with Example 1, any shuffling of a run in $\mathcal{B}_1$ with a run in $\mathcal{B}_2$ in which some time passage in a state of $\mathcal{B}_1$ mixes with some time passage in a state of $\mathcal{B}_2$. To this end, we will first duplicate all transitions in $\mathcal{B}$, such that each symbol $a \in \Sigma$ is

preceded by a duplicate symbol $\overline{a}$ in a copy of Σ. This is done by introducing, for each transition, a new state in which the control stays for 0 time units. The use of the new symbols $\overline{a}$ is the same as the extra symbols z_1, z_2 in Example 1. This process requires the introduction of a new clock y, which tests that each a is preceded in zero time by the corresponding $\overline{a}$.

Formally, we replace $\mathcal{B}$ with $\overline{\mathcal{B}} = (Q \cup \delta, \{x, y\}, \Sigma \cup \overline{\Sigma}, \overline{\eta}, \overline{\delta}, Q_0, Q_f)$ where

- For each $q \in Q$, $\overline{\eta}(q) = \{y\}$, and for each $q \xrightarrow{C,a,X} r \in \delta$, $\overline{\eta}(q \xrightarrow{C,a,X} r) = \eta(q)$.
- The transition relation is

$$\overline{\delta} = \left\{ q \xrightarrow{C,a,X\cup\{y\}} \tau, \tau \xrightarrow{(y=0),\overline{a},\emptyset} r \mid \tau = q \xrightarrow{C,a,X} r \in \delta \right\}$$

We straightforwardly have:

$$L(\overline{\mathcal{B}}) = \left\{ t_1\, a_1\, \overline{a}_1\, t_2\, a_2\, \overline{a}_2 \ldots t_n\, a_n\, \overline{a}_n \mid t_1\, a_1\, t_2\, a_2 \ldots t_n\, a_n \in L(\mathcal{B}) \right\}$$

Note also that $L(\mathcal{B}) = \lambda'(L(\overline{\mathcal{B}}))$ where $\lambda' : \Sigma \cup \overline{\Sigma} \to \Sigma \cup \{\varepsilon\}$ is the renaming defined by $\lambda'(a) = a$, $\lambda'(\overline{a}) = \varepsilon$ for all $a \in \Sigma$. Therefore our goal changes to finding automata $\mathcal{B}_1, \mathcal{B}_2$ and $\mathcal{B}_3$ such that $L(\overline{\mathcal{B}}) = \big(L(\mathcal{B}_1) \sqcup\!\sqcup L(\mathcal{B}_2)\big) \cap L(\mathcal{B}_3)$.

The first timed automaton is $\mathcal{B}_1 = \big(Q, \Sigma, \{x, x'\}, \delta_1, Q_0, Q_f\big)$ where δ_1 consists of the following tuples:

- $q \xrightarrow{C,a,X} r$, if both q and r are x-active or both x-inactive and $q \xrightarrow{C,a,X} r \in \delta$.
- $q \xrightarrow{C,a,X\cup\{x'\}} r$, if q is x-active and r is x-inactive and $q \xrightarrow{C,a,X} r \in \delta$.
- $q \xrightarrow{C\wedge(x'=0),a,X} r$, if q is x-inactive and r is x-active and $q \xrightarrow{C,a,X} r \in \delta$.

We may characterize the language of $\mathcal{B}_1$ as follows: $L(\mathcal{B}_1)$ contains a timed word w iff there exists a trajectory in $\mathcal{B}$, $\theta = \big((q_{j-1}, v_{j-1}) \xrightarrow{\zeta_j} (q_j, v_j)\big)_{1\le j\le k}$ and a decomposition of w as $w = \zeta'_1\zeta'_2 \ldots \zeta'_k$ such that

$$\zeta'_j = \begin{cases} \zeta_j & \text{if } \zeta_j \in \Sigma \text{ or } \Big(\zeta_j \in \mathbb{R} \text{ and } q_{j-1} \text{ is } x\text{-active, } \Big) \\ 0 & \text{otherwise} \end{cases}$$

The second timed automaton is $\mathcal{B}_2 = \big(\delta \cup Q_f, \overline{\Sigma}, \{y\}, \delta_2, Q_0^2, Q_f\big)$ where $Q_0^2 = \big\{ q \xrightarrow{C,a,X} r \in \delta \mid q \in Q_0 \big\}$ and the transition relation δ_2 is composed of the following tuples:

- $\tau \xrightarrow{true,\overline{a},\{y\}} \tau'$ where $\tau = q \xrightarrow{C,a,X} r$, $\tau' = r \xrightarrow{C',a',X'} s$ and q is x-inactive.
- $\tau \xrightarrow{(y=0),\overline{a},\emptyset} \tau'$ where $\tau = q \xrightarrow{C,a,X} r$, $\tau' = r \xrightarrow{C',a',X'} s$ and q is x-active.
- $\tau \xrightarrow{true,\overline{a},\emptyset} r$ where $\tau = q \xrightarrow{C,a,X} r$ with $r \in Q_f$ and q is x-inactive.
- $\tau \xrightarrow{(y=0),\overline{a},\emptyset} r$ where $\tau = q \xrightarrow{C,a,X} r$ with $r \in Q_f$ and q is x-active.

Similarly to the case of $\mathcal{B}_1$, we may characterize the language of $\mathcal{B}_2$ as follows: $L(\mathcal{B}_2)$ contains a timed word w iff there exists a trajectory in $\mathcal{B}$, $\theta = \big((q_{j-1}, v_{j-1}) \xrightarrow{\zeta_j} (q_j, v_j)\big)_{1\leq j\leq k}$ and a decomposition of as $w = \zeta'_1\zeta'_2\ldots\zeta'_k$ such that

$$\zeta'_j = \begin{cases} \overline{\zeta}_j & \text{if } \zeta_j \in \Sigma \text{ or } \Big(\zeta_j \in \mathbb{R} \text{ and } q_{j-1} \text{ is } x\text{-inactive } \Big) \\ 0 & \text{otherwise} \end{cases}$$

The third automaton is $\mathcal{B}_3 = (Q \cup \delta, \{y'\}, \Sigma \cup \overline{\Sigma}, \delta_3, Q_0, Q_f)$ where the transition relation is

$$\delta_3 = \Big\{ q \xrightarrow{\mathsf{true},a,\{y'\}} \tau, \tau \xrightarrow{(y'=0),\overline{a},\{y'\}} r \mid \tau = q \xrightarrow{C,a,X} r \in \delta \Big\}$$

The characterization of $L(\mathcal{B}_3)$ is the following: a word w belongs to $L(\mathcal{B}_3)$ iff there exists in $\mathcal{B}$ a trajectory $\theta = \big((q_{j-1}, v_{j-1}) \xrightarrow{\zeta_j} (q_j, v_j)\big)_{1\leq j\leq 2k}$ such that $w = \zeta'_1\zeta'_2\ldots\zeta'_{4k}$ where for all $1 \leq i \leq k$,

$$\zeta'_{4i-3} \in \mathbb{R}_{\geq 0} \qquad \zeta'_{4i-2} = \zeta_{2i}$$
$$\zeta'_{4i-1} = 0 \qquad \zeta'_{4i} = \overline{\zeta}_{2i}$$

It is then easy to note that

$$L(\overline{\mathcal{B}}) = \big(L(\mathcal{B}_1) \sqcup\!\sqcup L(\mathcal{B}_2)\big) \cap L(\mathcal{B}_3)$$

The Kleene theorem for timed automata [ACM97] ensures the existence of timed regular expressions equivalent with each of the three automata ($\mathcal{B}_1$. $\mathcal{B}_2$ and $\mathcal{B}_3$), fact which ends our proof. □

Since the equivalence between stopwatch automata and timed shuffle expressions is effective, we also get the following result:

Theorem 6. *The problem of checking for emptiness the semantics of a given timed shuffle expression is undecidable.*

The following result, which can be found in [Dim05], strengthens the undecidability theorem:

Theorem 7. *The emptiness problem for timed shuffle expressions without renaming is undecidable.*

The proof of this theorem is based on the fact that, similar to stopwatch automata, timed shuffle expressions without renaming may induce more general linear constraints than those induced by timed regular expressions. As an example, consider the following expression

$$E_{1/2} = \langle a\underline{\mathbf{t}}\hat{a}\, b\underline{\mathbf{t}}\hat{b}\rangle_1 c\underline{\mathbf{t}}\hat{c}\, d\underline{\mathbf{t}}\hat{d}\, e\underline{\mathbf{t}}\hat{e} \wedge \big(a\underline{\mathbf{t}}\hat{a}\langle b\underline{\mathbf{t}}\hat{b}\, c\underline{\mathbf{t}}\hat{c}\, e\underline{\mathbf{t}}\hat{e}\rangle_1 \sqcup\!\sqcup d\underline{\mathbf{t}}\hat{d}\big) \wedge$$
$$a\underline{\mathbf{t}}\hat{a}\, b\underline{\mathbf{t}}\hat{b}\langle c\underline{\mathbf{t}}\hat{c}\, d\underline{\mathbf{t}}\hat{d}\rangle_1 e\underline{\mathbf{t}}\hat{e} \wedge a\underline{\mathbf{t}}\hat{a}\, b\underline{\mathbf{t}}\hat{b}\, c\underline{\mathbf{t}}\hat{c}\langle d\underline{\mathbf{t}}\hat{d}\, e\underline{\mathbf{t}}\hat{e}\rangle_1 \quad (5)$$

Note that

$$\|E_{1/2}\| = \left\{ at_1\hat{a}\, bt_2\hat{b}\, ct_3\hat{c}\, dt_4\hat{d}\, et_5\hat{e} \mid \underline{t_1 = 2t_3}, t_1 + t_2 = 1 = t_3 + t_4, t_3 = t_5 \right\}$$

As this example suggests, time shuffle expressions without renaming are able to express division by 2, fact which is instrumental in proving the undecidability of the emptiness problem for stopwatch automata in [HKPV98].

We end this section with the following property:

Proposition 3. *Timed shuffle expressions without renaming are more expressive than timed regular expressions.*

Proof. We will prove that the semantics of the following expression cannot be accepted by a timed automaton:

$$E_0 = \langle a\underline{\mathbf{t}}\hat{a}b\underline{\mathbf{t}}\hat{b}\rangle_1 c\underline{\mathbf{t}}\hat{c} \wedge \left(\langle a\underline{\mathbf{t}}\hat{a}c\underline{\mathbf{t}}\hat{c}\rangle_1 \sqcup\!\sqcup b\underline{\mathbf{t}}\hat{b}\right)$$

We rely on the following Proposition (that is proved e.g. in [OW03,Dim03]) saying roughly that timed words having the same untiming and which satisfy the same timing constraints cannot be distinguished by timed automata. Here, $\lfloor \alpha \rfloor$ denotes the integral part and $\mathsf{frac}(\alpha)$ is the fractional part of the real number $\alpha \in \mathbb{R}$. Also, for a timed word $w = t_1 a_1 \dots t_n a_n$, we denote $w_{ij} = t_i a_i \dots t_j a_j$, and recall the notation $\ell(w) = \sum_{1 \le i \le n} t_i$.

Proposition 4. *Consider $w = t_1 a_1 \dots a_{n-1} t_n a_n$ and $w' = t'_1 a_1 \dots a_{n-1} t'_n a_n t'_{n+1}$, two timed words. Suppose further that for all $1 \le i \le j \le n$,*

$$\left\lfloor \ell(w_{ij}) \right\rfloor = \left\lfloor \ell(w'_{ij}) \right\rfloor \text{ and } \mathsf{frac}\big(\ell(w_{ij})\big) \neq 0 \text{ if and only if } \mathsf{frac}\big(\ell(w'_{ij})\big) \neq 0 \qquad (6)$$

Then, for any timed automaton $\mathcal{A}$, $w \in L(\mathcal{A})$ if and only if $w' \in L(\mathcal{A})$. Moreover, the two timed words are accepted along the same run in $\mathcal{A}$.

We may then apply this remark to the two timed words $w = a\,0.5\,\hat{a}\,b\,0.5\,\hat{b}\,c\,0.5\,\hat{c} \in \|E_0\|$ and $w' = a\,0.3\,\hat{a}\,b\,0.7\,\hat{b}\,c\,0.3\,\hat{c} \notin \|E_0\|$. Clearly, w and w' meet the condition (6). By contradiction, this implies that no timed automaton (and hence no timed regular expression) can be equivalent to E_0. □

Along the lines of Proposition 3, we may also prove that the timed language in Figure 3 is not timed regular: take $w_1 = 0.5a0.5a0.5a0.5a$ and $w_2 = 0.2a0.6a0.8a0.4a$, and observe that they both satisfy the conditions in Proposition 4, but $w_1 \in L(\mathcal{A}_3)$ and $w_2 \notin L(\mathcal{A}_3)$. Hence, $L(\mathcal{A}_3)$ is not timed regular.

As a corollary, partitioned stopwatch automata are strictly more expressive than timed automata.

3.1 Fair shuffle expressions

Definition 6. *The set of **fair shuffle expressions** is the subset of timed shuffle expressions defined recursively as follows:*

$$F ::= T \mid [a/b]F \mid F_1 + F_2 \mid F \wedge U \mid (F)^* \mid F_1 \sqcup\!\sqcup F_2$$

where T is a timed regular expression, U is an unconstrained expression and $a \in \Sigma$, $b \in \Sigma \cup \{\varepsilon\}$.

The following expression is a fair shuffle expression which is equivalent with the partitioned stopwatch automaton in Figure 3:

$$[z_1/\varepsilon, z_2/\varepsilon]\big((z_1\underline{\mathbf{t}}az_2\underline{\mathbf{t}}az_1\underline{\mathbf{t}}az_2\underline{\mathbf{t}}a) \wedge (\langle z_1\langle\underline{\mathbf{t}}\rangle_{]0,1[}az_1\underline{\mathbf{t}}a\rangle_1) \sqcup\!\sqcup (\langle z_2\langle\underline{\mathbf{t}}\rangle_{]0,1[}z_2\underline{\mathbf{t}}a\rangle_1)\big)$$

Theorem 8. *Partitioned stopwatch automata are equivalent with fair shuffle expressions, and the equivalence is effective.*

Proof. For the left-to-right inclusion, take $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \eta, \delta, Q_0, Q_f)$ some partitioned stopwatch automaton. We partition the set of states Q into subsets $S_1, \ldots, S_n$ such that $\eta(q) = \eta(q')$ iff $q, q' \in S_i$, for some i. We also put $\mathcal{X}_i = \eta(S_i)$. By Lemma 1, we can assume that if $q \xrightarrow{C,a,Y} q' \in \delta$, then $Y \subseteq \eta(q')$ and $C \in \mathsf{Constr}(\eta(q'))$, and that the final states have no outgoing transitions.

The idea is to construct n timed automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$ such that each $\mathcal{A}_i$ copies all the states and performs all the actions of $\mathcal{A}$, but lets time pass only in states from S_i, and allows only zero time passage in the other states. The languages of all $\mathcal{A}_i$ will be shuffled, and intersected with the language of a zero-constrained timed automaton that will ensure proper interleaving. That is, the connecting property between these automata is the following:

$$L(\mathcal{A}) = f\Big(\big(L(\mathcal{A}_1) \sqcup\!\sqcup \ldots \sqcup\!\sqcup L(\mathcal{A}_n)\big) \cap L(\mathcal{A}')\Big)$$

As in the proof of Theorem 5, we will assume that all transitions in δ are labeled with distinct symbols, i.e., if $q \xrightarrow{C,a,X} r, q' \xrightarrow{C',a',X'} r' \in \delta$ and $a = a'$ then $q = q', r = r', C = C', X = X'$. Each timed automaton will also reference a new clock x_i which is needed for ensuring that time passage is 0 in each automaton $\mathcal{A}_i$ while passing through a state q with $q \notin S_i$.

Since each action in Σ is to be executed in each automaton, the renaming f is meant to delete $n-1$ copies of each action and keep only the n-th. This can be done by modifying the set of symbols utilized by each $\mathcal{A}_i$ to a distinct copy $\Sigma_i = \Sigma \times \{i\}$ of the intial set of symbols, and, for each $a \in \Sigma$, putting $f(a, 1) = a$ and $f(a, i) = \varepsilon$, for all $2 \leq i \leq n$. The set of symbols used by $\mathcal{A}'$ will be the union of all these copies of Σ.

Formally, $\mathcal{A}_i = (Q, \Sigma_i, \mathcal{X}_i, \delta_i, Q_0, Q_f)$ with

- $\Sigma_i = \Sigma \times \{i\}$.
- $\mathcal{X}_i = \eta(S_i) \cup \{x_i\}$, where $x_i \notin \mathcal{X}$ is a new distinct clock.
- δ is composed of the following tuples:
  - $q \xrightarrow{C|_{\mathcal{X}_i},a,X\cap\mathcal{X}_i} r$ for each $q \xrightarrow{C,a,X} r \in \delta$ with $q \in S_i$ and $r \notin Q_f$.
  - $q \xrightarrow{C|_{\mathcal{X}_i}\wedge(x_i=0),a,(X\cap\mathcal{X}_i)\cup\{x_i\}} r$ for each $q \xrightarrow{C,a,X} r \in \delta$ with $q \notin S_i$ and $r \notin Q_f$.
  - $q \xrightarrow{C|_{\mathcal{X}_i},a,\emptyset} r$ for each $q \xrightarrow{C,a,X} r \in \delta$ with $q \in S_i$ and $r \in Q_f$.
  - $q \xrightarrow{C|_{\mathcal{X}_i}\wedge(x_i=0),a,\emptyset} r$ for each $q \xrightarrow{C,a,X} r \in \delta$ with $q \notin S_i$ and $r \in Q_f$.

The intersection automaton is the following: $\mathcal{A}' = (Q', \Sigma', \{x'\}, \delta', Q'_0, Q'_f)$ in which

– $Q' = \delta \times 2^{\{1,\ldots,n\}}$.
– $Q'_0 = \big\{(q \xrightarrow{C,a,X} r, \emptyset) \mid q \xrightarrow{C,a,X} r \in \delta \text{ with} q \in Q_0\big\}$.
– $Q'_f = \big\{q \xrightarrow{C,a,X} r, \{1,\ldots,n\} \mid q \xrightarrow{C,a,X} r \in \delta, r \in Q_f\big\}$.
– $\Sigma' = \bigcup_{1\leq i\leq n} \Sigma_i$.
– δ' is composed of the following transitions:

$$\begin{aligned}\delta' = \big\{(q \xrightarrow{C,a,X}, A) \xrightarrow{x'=0,(a,i),\{x'\}} (q \xrightarrow{C,a,X} r, B)) \mid q \xrightarrow{C,a,X} r \in \delta, i \notin A, A' = A \cup \{i\}\big\} \\ \cup \big\{q \xrightarrow{C,a,X}, \{1,\ldots,n\}) \xrightarrow{true,\varepsilon,\{x'\}} (r \xrightarrow{C',b,X'} s, \emptyset)) \mid q \xrightarrow{C,a,X} r, r \xrightarrow{C',b,X'} s \in \delta\big\}\end{aligned}$$

As a consequence of Theorem 5, we can construct timed regular expressions $T_1, \ldots, T_n$ such that $\|T_i\| = L(\mathcal{A}_i)$ for all $1 \leq i \leq n$. Furthermore, since $\mathcal{A}'$ is a zero-constrained timed automaton, we can construct an unconstrained expression T_0 such that $\|T_0\| = L(\mathcal{A}')$. We then have that

$$L(\mathcal{A}) = f(T_1 \sqcup\!\sqcup \ldots \sqcup\!\sqcup T_n) \wedge T_0)$$

which ends the proof of the left-to-right inclusion.

For the right-to-left inclusion, we proceed by structural induction on the given regular expression T. The case when T is a timed regular expression is already covered by Theorem 5 while the cases of union, concatenation, star, shuffle and renaming can be treated exactly as in the proof of the Theorem 5, by observing that the resulting stopwatch automata are partitioned.

For the intersection case, suppose $T = T' \wedge U$, with T' a fair shuffle expression and U an unconstrained expression. Then, by induction, there exist $\mathcal{A}_1$ a partitioned stopwatch automaton for which $L(\mathcal{A}_1) = \|T'\|$. On the other hand, by Proposition 2, there exists a zero-constrained timed automaton $\mathcal{A}_2$ such that and $L(\mathcal{A}_2) = \|U\|$. Denote both automata as $\mathcal{A}_1 = (Q_1, \Sigma, \mathcal{X}_1, \eta_1, \delta_1, Q_0^1, Q_1^f)$, resp. $\mathcal{A}_2 = (Q_2, \Sigma, \{x\}, \delta_2, Q_0^2, Q_2^f)$.

The intersection construction between $\mathcal{A}_1$ and $\mathcal{A}_2$ will then work on pairs of states in $Q_1 \times Q_2$. The clock x of $\mathcal{A}_2$ cannot be transformed into a stopwatch active in every state, since in the resulting stopwatch automaton would not be partitioned. Therefore, for each $q \in Q_1$ we add a new stopwatch x_q (as a copy of x) which will be reset when entering in any state in the intersection automaton which is of the form (q, q') for some $q' \in Q_2$. All these stopwatches will then be grouped, according to the partition of Q_1 inherited from $\mathcal{A}_1$, into classes of states that have the same set of active stopwatches.

Formally, we construct the automaton $\mathcal{A} = (Q_1 \times Q_2, \Sigma, \mathcal{X}_1 \cup \mathcal{X}_2, \eta, \delta, Q_0^1 \times Q_0^2, Q_f^1 \times Q_f^2)$ in which $\mathcal{X}_2 = \{x_q \mid q \in Q_1\}$, $\eta(q, q') = \eta_1(q) \cup \{x_r \mid \eta(q) = \eta(r)\}$ and

$$\begin{aligned}\delta = \big\{(q_1, q_2) \xrightarrow{C\wedge(x_{q_1}\in I),a,X\cup\{x_{q_1}\}} (q_1', q_2') \mid q_1 \xrightarrow{C,a,X} q_1' \in \delta_1, q_2 \xrightarrow{x\in I,a,\{x\}} q_2' \in \delta_2\big\} \\ \cup \big\{(q_1, q_2) \xrightarrow{C,\varepsilon,X\}} (q_1', q_2') \mid q_1 \xrightarrow{C,\varepsilon,X} q_1' \in \delta_1, q_2 \in Q_2\big\} \\ \cup \big\{(q_1, q_2) \xrightarrow{x_{q_1}\in I,\varepsilon,X} (q_1', q_2') \mid q_1 \in Q_1, q_2 \xrightarrow{x\in I,\varepsilon,\{x\}} q_2' \in \delta_2\big\}\end{aligned}$$

The last two types of transitions are needed since ε-transitions can be executed in one of the automata asynchronously from ε-transitions in the second automaton.

We may then prove that $\Big((q_{i-1}, q'_{i-1}), v_{i-1}) \xrightarrow{\xi_i} ((q_i, q'_i), v_i)\Big)_{1\le i\le k}$ is a trajectory of $\mathcal{A}$ iff $\Big((q_{i-1}, (v_{i-1}|_{\chi_1})) \xrightarrow{\xi_i} (q_i, (v_i|_{\chi_1}))\Big)_{1\le i\le k}$ is a trajectory of $\mathcal{A}_1$ and $\Big((q'_{i-1}, u_{i-1}) \xrightarrow{\xi_i} (q'_i, u_i)\Big)_{1\le i\le k}$ is a trajectory of $\mathcal{A}_1$, where $u_i(x) = v_i(x_{q_i})$. Obviously this implies that $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$.

This ends the proof of the reverse inclusion. □

Corollary 3. *The class of partitioned stopwatch automata is closed under union, renaming, shuffle, Kleene star.*

In the next section we will prove that partitioned stopwatch automata are not closed under intersection.

4 Distributed time-asynchronous automata

We present here the class of distributed time-asynchronous automata, which are synchronous compositions of timed automata in which local times may pass independently in different components. The class presented here differs from the class in [DL07] by the fact that the components do not utilized generalized resets.

Definition 7. *A **distributed time-asynchronous automaton** is a tuple*

$$\mathcal{A} = (Q_1, \ldots, Q_n, \Sigma, \mathcal{X}_1, \ldots, \mathcal{X}_n, \delta_1, \ldots, \delta_n, Q_0^1, \ldots, Q_0^n, Q_f^1, \ldots Q_f^n)$$

where

- Σ *is a finite set of* symbols.
- $\mathcal{X}_1, \ldots, \mathcal{X}_n$ *are* n *finite, pairwise-disjoint sets of* clocks. *We denote in the sequel* $\mathcal{X} = \bigcup_{1\le i\le n} \mathcal{X}_i$ *and call* $\mathcal{X}_i$ *the set of clocks* owned by *component* i.
- $Q_1, \ldots, Q_n$ *are* n *finite sets of* locations. Q_i *is the set of locations of component* i.
- $Q_0^i \subseteq Q_i$ *is the set of* initial locations *in component* i, *and* $Q_f^i \subseteq Q_i$ *is its set of* final *locations* ($1 \le i \le n$).
- *And* $\delta_1, \ldots, \delta_n$ *are* transition relations, *with* $\delta_i \subseteq \big\{q \xrightarrow{C,\xi,X} r \mid q, r \in Q_i, \xi \in \Sigma \cup \{\varepsilon\}, C \in \mathsf{Constr}\big(\bigcup_{1\le i\le n} \mathcal{X}_i\big), X \subseteq \mathcal{X}_i\big\}$.

Note that a timed automaton is a distributed time-asynchronous automaton with only one component.

Intuitively, a distributed time-asynchronous automaton can make time-passage transitions in which clocks in different components evolve independently, discrete transitions in which components may synchronize, and internal, silent transitions executed in one component independently of the other components.

In discrete, synchronizing transitions, each component checks the validity of a clock constraint (that may refer to any clocks in $\mathcal{X}$) and, upon validity, all agree on the same symbol $a \in \Sigma$ and reset some clocks while changing location. Any component may reset only the clocks that it owns, but any component may read clocks not owned by it.

In silent transitions, a specified component checks for the validity of a clock constraint, then resets some clocks it owns and changes location "silently", i.e. on an ε-transition. Such a transition might be executed by a component without requiring any synchronization with other components; hence, only the control location of the component that executed the silent transition changes, the other components might keep their control location unchanged.

Formally, the semantics of a distributed time-asynchronous automaton is a *timed transition system* $\mathcal{T}(\mathcal{A}) = (\mathcal{Q}, \theta, \mathcal{Q}_0, \mathcal{Q}_f)$ where:

$$\mathcal{Q} = Q_1 \times \ldots \times Q_n \times [\mathcal{X} \to \mathbb{R}_{\geq 0}] \tag{7}$$

$$\mathcal{Q}_0 = Q_0^1 \times \ldots \times Q_0^n \times \{\mathbf{0}_{\mathcal{X}}\} \tag{8}$$

$$\mathcal{Q}_f = Q_f^1 \times \ldots \times Q_f^n \times [\mathcal{X} \to \mathbb{R}_{\geq 0}] \tag{9}$$

$$\begin{aligned}\theta = \big\{ (q_1, \ldots, q_n, v) \xrightarrow{t} (q_1, \ldots, q_n, v') \mid \text{ for all } 1 \leq i \leq n, q_i \in Q_i \text{ and} \\ \text{there exists } t_i \in \mathbb{R}_{\geq 0} \text{ with } v'\big|_{\mathcal{X}_i} = v\big|_{\mathcal{X}_i} + t_i \text{ and } t = t_1 + \ldots + t_n \big\}\end{aligned} \tag{10}$$

$$\begin{aligned}\cup \big\{ (q_1, \ldots, q_n, v) \xrightarrow{a} (q_1', \ldots, q_n', v') \mid a \in \Sigma \text{ and for all } i \leq n \text{ there exists } C_i \in \mathsf{Constr}(\mathcal{X}) \\ \text{and } X_i \subseteq \mathcal{X}_i \text{ with } q_i \xrightarrow{C_i, a, X_i} q_i' \in \delta_i, v \models C_i \text{ and } v' = v[X_1 \cup \ldots \cup X_n := 0] \big\}\end{aligned} \tag{11}$$

$$\begin{aligned}\cup \big\{ (q_1, \ldots, q_n, v) \xrightarrow{\varepsilon} (q_1', \ldots, q_n', v') \mid \text{ there exist } 1 \leq i \leq n, C_i \in \mathsf{Constr}(\mathcal{X}), \text{ and } X_i \subseteq \mathcal{X}_i \\ \text{with } v \models C_i, q_i \xrightarrow{C_i, \varepsilon, X_i} q_i' \in \delta_i, v' = v[X_i := 0] \text{ and } q_j' = q_j \text{ for all } j \neq i \big\}\end{aligned} \tag{12}$$

A *trajectory* in $\mathcal{A}$ is a sequence of transitions in θ, alternating between time-passage transitions and discrete transitions:

$$trj = \big((q_1^{j-1}, \ldots, q_n^{j-1}, v_{j-1}) \xrightarrow{\xi_j} (q_1^j, \ldots, q_n^j, v_j) \big)_{1 \leq j \leq 2m}$$

that starts in the initial states of $\mathcal{T}(\mathcal{A})$, with $\xi_{2j-1} \in \mathbb{R}_{\geq 0}$ and $\xi_{2j} \in \Sigma \cup \{\varepsilon\}$ for all $1 \leq j \leq m$. The trajectory trj is *accepting* if it ends in $\mathcal{Q}_f$ and *ends with a synchronization transition* of the type 11 above. The *timed word accepted by* trj is $\mathsf{acc}(trj) = \xi_1 \ldots \xi_m$. The *timed language accepted by* $\mathcal{A}$ is $L(\mathcal{A}) = \{acc(trj) \mid trj \text{ is accepted by } A\}$.

Remark 5. Note that we request that all components execute a synchronization transition when accepting a timed word, hence the final symbol in each accepted word must be a symbol in Σ.

An example of a distributed time-asynchronous automaton is given in Figure 7 below. The language accepted by this automaton is the same as the language accepted by the partitioned stopwatch automaton in Figure 3.

Before going to the proof of the equivalence of distributed time-asynchronous automata with partitioned stopwatch automata, let us also adapt the notion of *run* to the case of distributed time-asynchronous automata. First, a *run in component* i of a distributed time-asynchronous automaton $\mathcal{A}$ is a sequence $\rho_i = (q_{j-1}^i \xrightarrow{C_j^i, \xi_j, X_j^i} q_j^i)_{1 \leq j \leq m}$ consisting of transitions in δ_i or tuples of the form $q_{i-1} \xrightarrow{\mathsf{true}, \varepsilon, \emptyset} q_i$, which we call *idle transitions* – their utility will be apparent in the

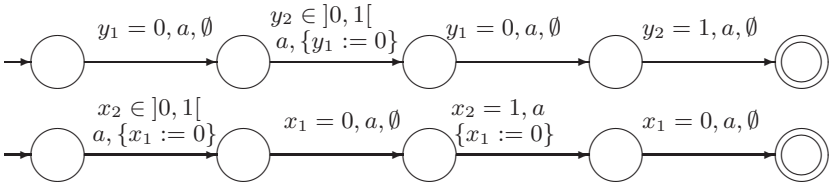

Fig. 7. The two components of a distributed time-asynchronous automaton recognizing the language $\{t_1\, a\, t_2\, a\, t_3\, a\, t_4\, a \mid t_1, t_2, t_3, t_4 \in]0,1[\,, t_1 + t_3 = 1 \wedge t_2 + t_4 = 1\}$

sequel. A *run in the distributed time-asynchronous automaton* $\mathcal{A}$ is then a tuple $\overline{\rho} = (\rho_i)_{1\leq i\leq n}$ such that $\rho_i = (q^i_{j-1} \xrightarrow{C^i_j,\xi_j,X^i_j} q^i_j)_{1\leq i\leq m}$, is a run in component i and the sequence of symbols $(\xi_1, \xi_2, \ldots, \xi_m)$ is the same in all runs.

Given a run in $\mathcal{A}$, $\overline{\rho} = (\rho_i)_{1\leq i\leq n}$, with $\rho_i = (q^i_{j-1} \xrightarrow{C^i_j,\xi_j,X^i_j} q^i_j)_{1\leq i\leq m}$, and a trajectory $trj = \big((q^1_{j-1}, \ldots, q^n_{j-1}, v_{j-1}) \xrightarrow{\overline{\xi}_j} (q^1_j, \ldots, q^n_j, v_j)\big)_{1\leq j\leq 2m}$, we say that trj is *associated with* $\overline{\rho}$ if the following properties hold:

1. For each $1 \leq j \leq m$, $\overline{\xi}_{2j} \in \Sigma \cup \{\varepsilon\}$ and $\overline{\xi}_{2j-1} \in \mathbb{R}_{\geq 0}$.
2. The sequence of actions is the same in both $\overline{\rho}$ and trj, that is, for all i, $\xi_1\xi_2 \ldots \xi_m = \overline{\xi}_2\overline{\xi}_4 \ldots \overline{\xi}_{2m}$.
3. For each $1 \leq j \leq m$ and $1 \leq i \leq n$, $v_{2j-1} \models C^i_j$ and $v_{2j} = v_{2j-1}[X^1_j \cup \ldots \cup X^n_j := 0]$.

Related with the above definition, note that, for each $1 \leq j \leq m$, $\overline{\xi}_{2j-1}$ can be decomposed as $\overline{\xi}_{2j-1} = t^1_{2j-1} + \ldots + t^n_{2j-1}$ such that for all i, $v_{2j-1}\big|_{\mathcal{X}_i} = v_{2j-2}\big|_{\mathcal{X}_i} + t^i_{2j-1}$. We then say that the timed word $t^i_1\overline{\xi}_2 t^i_3\overline{\xi}_4 \ldots t^i_{2m-1}\overline{\xi}_{2m}$ is *associated with* ρ_i.

Remark 6. Some comments are in order here on the definition of runs in components and their association with trajectories in distributed time-asynchronous automaton. The utilization of idle transitions in the runs ρ_i is just a technical trick that allows a simpler definition of runs in distributed time-asynchronous automaton and their association with trajectories. The fact that some instant $2j$ along a trajectory trj is associated with one ε-transition in each component (signaled by the transition $(q_{2j-1}, v_{2j-1}) \xrightarrow{\xi_{2j}=\varepsilon} (q_{2j}, v_{2j})$) does not mean that the components synchronize at that moment by some "hidden" communication mechanism, since some of these ε-transitions might simply be the idle transitions $q \xrightarrow{\mathsf{true},\varepsilon,\emptyset} q$. This association just avoids heavy extra notations for the positions where visible, synchronization-used symbols occur in each ρ_i.

Theorem 9. *The classes of distributed time-asynchronous automata and partitioned stopwatch automata are equivalent over the class of timed languages $\mathcal{L}$ with the following properties:*

$$L \in \mathcal{L} \text{ iff } \forall w \in L, w = w'a \text{ with } a \in \Sigma$$

Proof. For the left-to-right inclusion, we will simply gather together all the states of the components of a distributed time-asynchronous automaton into a single centralized control, and transform all clocks into stopwatches. Each state q belonging to some component i will continue to own the stopwatches coming from that component, which will mean that only those stopwatches

will be incremented during time passage in q. Hence, the clock partition coming from the original distributed time-asynchronous automaton will give the stopwatch partition in the resulting stopwatch automaton.

For ensuring that each accepting trajectory ends with a symbol in Σ, we assume w.l.o.g. that final states in each of the components of $\mathcal{A}$ cannot be reached through ε-transitions. This can be easily achieved using state-splitting techniques.

Formally, starting with a given distributed time-asynchronous automaton

$$\mathcal{A} = (Q_1, \ldots, Q_n, \Sigma, \mathcal{X}_1, \ldots, \mathcal{X}_n, \delta_1, \ldots, \delta_n, Q_0^1, \ldots, Q_0^n, Q_f^1, \ldots Q_f^n),$$

we construct the partitioned stopwatch automaton $\mathcal{A}' = (Q, \Sigma, \mathcal{X}, \eta, \delta, Q_0, Q_f)$ in which:

- $Q = Q_1 \times \ldots \times Q_n \times \{1, \ldots, n\}$ where the last item of each tuple represents the component for which the time is allowed to elapse.
- The set of clocks is $\mathcal{X} = \bigcup_{i \in [1,n]} \mathcal{X}_i$ and η is such that $\eta(q_1, \ldots, q_n, j) = \mathcal{X}_j$.
- δ is the following set of transitions:

$$\begin{aligned}
\delta = \Big\{ & (q_1, \ldots, q_n, j) \xrightarrow{C,a,Y} (q_1', \ldots, q_n', j) \mid \text{ for all } 1 \leq i \leq n \text{ there exists } q_i \xrightarrow{C_i,a,Y_i} q_i' \in \delta_i \\
& \qquad \text{with } a \in \Sigma, Y = \bigcup\nolimits_{1 \leq i \leq n} Y_i \text{ and } C = \bigwedge\nolimits_{1 \leq i \leq n} C_i \Big\} \\
& \cup \Big\{ (q_1, \ldots, q_i, \ldots, q_n, i) \xrightarrow{C,\varepsilon,Y} (q_1', \ldots, q_i', \ldots, q_n', i) \mid q_i \xrightarrow{C,\varepsilon,Y} q_i' \in \delta_i \text{ and } q_j = q_j' \forall j \neq i \Big\} \\
& \cup \Big\{ (q_1, \ldots, q_n, j) \xrightarrow{\mathsf{true},\epsilon,\emptyset} (q_1, \ldots, q_n, j+1)) \mid 1 \leq j < n \Big\} \\
& \cup \Big\{ (q_1, \ldots, q_n, n) \xrightarrow{\mathsf{true},\epsilon,\emptyset} (q_1, \ldots, q_n, 1) \Big\}
\end{aligned}$$

- $Q_0 = Q_0^1 \times \ldots \times Q_0^n \times \{1, \ldots, n\}$ and $Q_f = Q_f^1 \times \ldots \times Q_f^n \times \{1, \ldots, n\}$.

It is then straightforward to prove that $L(\mathcal{A}) = L(\mathcal{A}')$.

For the reverse inclusion, the main idea is to distribute the state space to n components, where n is the number of distinct classes of stopwatches. Note that we cannot use renaming here, as we *don't want* to prove that partitioned stopwatch automata and distributed time-asynchronous automata are equivalent *modulo renaming*.

The distribution will proceed as follows: first, each component will be in charge of exactly one class of the stopwatches of the original automaton $\mathcal{A}$ – which will be referred to as (classes of) clocks in the sequel. At each moment, only one component may increment its clocks – this component is called the *active component* – and all the other (*inactive*) components will have to update their information about the state in which the active component is before and after each discrete transition, or to take into account the fact that the active component is different after the transition. This information update has to be cross-checked for correctness by all components: the distributed time-asynchronous automaton proceeds if and only if the active component before a transition is ensured that the active component after that transition will start its execution in the appropriate state and all the other components remain inactive.

The cross-checks are implemented by means of a communication mechanism between components that involves resetting some clock to zero in the active component, while the inactive

components check that the respective clock is zero in order to ensure consistency. We explain first the communication mechanism for the simulation of transitions in δ for which all associated discrete transitions in $\mathcal{T}(\mathcal{A})$ that occur along an accepting trajectory are preceded and followed, on that trajectory, by non-zero delay transitions.

In this case, communications are "handshakes":

1. The mechanism is launched by the active component which decides to take a transition after a non-zero time passage in the source location of that transition. The decision of the active component consists of the duration of staying in the source location, the symbol to be executed and the identity of the component that will be activated after the transition. The state-region determinization construction from Subsection 2.4 ensures that these elements of the decision *uniquely identify* the transition to be taken in δ.
   We will use the term *source component* to identify the component that owns the stopwatches of the source state in this (unique) transition.
2. The source component "broadcasts" its decision by means of resetting some particular clock between an extra set of clocks that is assigned to that component. This clock is the unique clock that can be zero at the respective moment, between all the extra clocks assigned to the active component. The transition that represents this broadcast is an ε-transition that is executed by the source component asynchronously from other components.
3. With a zero delay after this ε-transition, the source component executes the simulated transition, which, when this simulated transition is labeled with some symbol $a \neq \varepsilon$, triggers the execution of a transition labeled with the same symbol in all the other components.
4. For the whole distributed time-asynchronous automaton to continue to work, the component that corresponds to the target state of that transition (call it the *target component*) will have to start working (or *get awake*) on the transition that is synchronized on symbol a. To this end, on this synchronizing transition, the target component detects the clock that was reset by the source component, and checks that its local location is identical with the source location of the transition desired by the source component. Then, it will let a short amount of time to pass, and, after that, reset, on an ε-transition, another specific clock to zero. Again, in the target component, this specific clock will be the unique clock owned by the component which has a zero value.
5. The source component gets the "acknowledgment" that the target component has started working by checking that this second specific clock is zero while all the other clocks of the target component are non-zero – again by employing an asynchronous ε-transition. This ε-transition will turn "asleep" the source component, resetting all its extra clocks and putting it a *transient* location – that is, each transitions leaving that location would have to check that the component spent zero time units in that location.
6. Finally, the target component will test (on an ε-transition too) that the source component has correctly reached its "sleeping" state, which ends the handshake and permits the target component to continue its work.
7. Additionnaly, all the other inactive components will have to remain asleep (a fact which is checked by both source and target component). They are also endowed with discrete a-transitions that allow them to consistently copy, in their location, the evolution of the simulated location of the given partitioned stopwatch automaton.

The above mechanism works almost identically for the simulation of ε-transitions, because, by state-region determinism, any ε-transition that occurs in an accepting trajectory must be separated from the previous and the next discrete transition by a non-zero delay, property which gives the possibility for both the source and the target components to achieve their communication via clock resets and small time passages. The difference is that all the components no longer have the possibility to synchronize on the simulated ε-transition – which may be executed asynchronously. But the "handshake" mechanism is designed such that the source component can only go into the sleeping state only after the target component executed its asynchronous copy of the simulated ε-transition.

The only problems that may occur in this case concern the other inactive components, since neither the source nor the target component have the possibility to check that the other inactive components have consistently simulated the current ε-transition of the given partitioned stopwatch automaton. (The source and the target component can still check that the other components are sleeping during the "handshake" process.) But this does not cause any simulation problems since, as requested at point 4 above, when, in the rest of the computation, an inactive component is to become active, it can only do so when being in the source state of the source component – which means that, if it missed at a previous moment its possibility to consistently record the simulated location, then the whole distributed time-asynchronous automaton will be blocked.

There is still a third situation that has to be simulated differently: it's the possibility to have sequences of discrete transitions all taken within an interval of length zero. This is where the last property of state-region determinism plays its role: recall that, by condition 5 in Definition 4, such sequences of discrete transitions separated by zero delays have the property that their source locations are all labeled with the same set of stopwatches – that is, in our terms, their source component is the same. Furthermore, by condition 4 in the same definition, none of these transitions are labeled with ε – and hence they will force synchronizations with the inactive components. And finally, by condition 2, the source component is *deterministic* on such sequences of transitions.

In this case, the handshake mechanism is the following:

1. The active component decides to take, after some non-zero time passage in some location, a sequence of discrete transitions in zero time. The decision of the active component consists of the duration of staying in the source location, the first symbol to be executed and the identity of the component that will be activated after the transition. The state-region determinization construction from Subsection 2.4 ensures that these elements of the decision *uniquely identify* the sequence of transitions to be taken in δ within an interval of zero length.
2. The source component "broadcasts" its decision by means of resetting, on an ε-transition, some particular clock between an extra set of clocks that is assigned to that component.
3. The source component executes all the simulated transitions, with a zero delay after this ε-transition and with zero delays between each transition. This also triggers the execution of the same sequence of transitions in all the other components – due to state-region determinism and the necessity to execute synchronously each visible transition in all components.
4. The target component of the last transition in this sequence (which is the only transition in this sequence that may change the active component) gets awake on the first transition, detects

the extra clock that was reset by the source component (and hence identifies which is the last transition in the sequence) and checks that its local state is identical with the source state of the transition desired by the source component. At the end of the sequence of transitions, it will let a short amount of time to pass, and, after that, reset, on an ε-transition, another specific clock to zero.

Note that, if the target component misleadingly lets time pass on a symbol that is not the last in this sequence of transitions, the whole partitioned stopwatch automaton will be blocked.

5. The source component gets the "acknowledgment" that the target component has started working by checking that this second specific clock is zero while all the other clocks of the target component are non-zero – again by employing an asynchronous ε-transition. This ε-transition puts the source component into a "sleeping" state.
6. Finally, the target component will test (on an ε-transition too) that the source component has correctly reached its "sleeping" state, which ends the handshake and permits the target component to continue its work.

So suppose we start with a state-region deterministic partitioned stopwatch automaton $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \eta, \delta, Q_0, Q_f)$, and $(\mathcal{X}_i)_{1 \leq i \leq n}$ denotes the partition of the set of stopwatches. The distributed time-asynchronous automaton equivalent with $\mathcal{A}$ is:

$$\mathcal{B} = \left(Q_1, \ldots, Q_n, \Sigma, \tilde{\mathcal{X}}_1, \ldots, \tilde{\mathcal{X}}_n, \delta_1, \ldots, \delta_n, Q_0^1, \ldots Q_0^n, Q_f^1, \ldots, Q_f^n\right)$$

where

1. The set of states of component i is

$$Q_i = (Q \times \{i\}) \cup (\delta \times \{i\}) \cup (\delta \times \delta \times \{i\}) \cup (Q \times \delta \times \{j \mid j \leq n, j \neq i\} \times \{i\}) \\ \cup (Q \times \delta \times \{i\}) \cup (Q \times Q \times \delta \times \{i\}) \cup (Q \times \{0\} \times \{i\}).$$

   Intuitively, a state (q, i) should allow nonzero time passage only if $\eta(q) = \mathcal{X}_i$ – these are the *active states*. A state (τ, i) with $\tau \in \delta$ is a transient state which occurs both in source component and in target component when the source component decides to take transition τ. States of the form $(q \xrightarrow{C,\xi,X} r, j, i)$ with $\eta(r) = \mathcal{X}_i$ are states used by component i to acknowledge to component $j \neq i$ with $\eta(q) = \mathcal{X}_j$ that it is awake and takes over the simulation of $\mathcal{A}$. Such states are persistent, and the only possibility to exit such states is for components i and j to accomplish their "handshake" mechanism described above. States of the form $(\tau, \overline{\tau}, i)$ are transient states in which the active component enters when deciding to take, in zero time, a sequence of transitions that starts with τ and ends with $\overline{\tau}$. States of the form (q, τ, i) are used either by the active component or by the inactive components during sequences of transitions taken in zero time. States of the form (q, r, τ, i) with $\eta(q) = \eta(r) = \mathcal{X}_i$ are used at the end of a sequence of transitions taken in zero time by both the active component and the component that will become active after this sequence of transitions. Finally, states of the form $(q, 0, i)$ are used in each component at the beginning of each trajectory that starts with a sequence of transitions taken in zero time.
2. $Q_0^i = (Q_0 \setminus \{q_0\} \times \{i\}) \cup \{(q_0, 0, i)\}$ and $Q_f^i = Q_f \times \{i\}$.

3. $\tilde{\mathcal{X}}_i = \mathcal{X}_i \cup \overline{\mathcal{X}}_i$ where $\overline{\mathcal{X}}_i$ denotes the set of extra clocks needed by component i,

$$\overline{\mathcal{X}}_i = \{y_i\} \cup \{x_{\tau,i} \mid \tau \in \delta\}$$

The clocks y_i are used for checking for non-zero time passage in component i, hence being reset after simulating each transition whose target state is component i. Clocks $x_{\tau,i}$ are used for the communications betweeen components: they are reset by the source component of transition τ to signal to all the other components that it will take transition τ. They are also reset by the target component in τ to signal to the source component that it got awake.
We also denote $\overline{\mathcal{X}}_{\mathcal{B}} = \bigcup_{1 \leq i \leq n} \overline{\mathcal{X}}_i$.

4. δ_i consists of the following transitions (we give the informal description of the transitions in the parentheses):

   (a) $(q,i) \xrightarrow{C',\varepsilon,\{x_{\tau,i}\}} (\tau,i)$ for $\tau = q \xrightarrow{C,\xi,X} r \in \delta$ with $\eta(q) = \mathcal{X}_i$ and

   $$C' = C \wedge (y_i > 0) \wedge \bigwedge_{l \neq i} \bigwedge_{x \in \mathcal{X}_l} (x = 0)$$

   (active component broadcasts intention to take transition τ). After such a transition is taken, between all clocks in $\tilde{\mathcal{X}}_i$ only clock $x_{\tau,i}$ equals zero. Also note that all the extra clocks in $\mathcal{X}_{\mathcal{B}} \setminus \tilde{\mathcal{X}}_i$ must be zero, which should signal the fact that component i is the only "active" component.

   (b) $(\tau,i) \xrightarrow{(x_{\tau,i}=0) \wedge (y_i > 0), a, X \cup \overline{\mathcal{X}}_i} (r,i)$ for $\tau = q \xrightarrow{C,a,X} r \in \delta$ with $\eta(q) = \eta(r) = \mathcal{X}_i$ (transition τ is executed in component i when target state is owned by the same component).

   (c) $(q,i) \xrightarrow{C,a,X \cup \overline{\mathcal{X}}_i} (\tau,i)$ for $\tau = q \xrightarrow{C',a,X} r \in \delta$ where $a \in \Sigma$, $\eta(q) = \mathcal{X}_j$, $\eta(r) = \mathcal{X}_i$ with $i \neq j$, and

   $$C = C' \wedge (x_{\tau,j} = 0) \wedge \bigwedge_{x \in \overline{\mathcal{X}}_j, x \neq x_{\tau,j}} (x > 0) \wedge \bigwedge_{l \neq j} \bigwedge_{x \in \overline{\mathcal{X}}_l} (x = 0)$$

   (target component in transition τ gets awaken, case of discrete visible transitions between distinct components). Note that by Lemma 2, we have that $X \subseteq \eta(r)$.

   (d) $(\tau,i) \xrightarrow{(x_{\tau,i}=0) \wedge \bigwedge_{x \in \overline{\mathcal{X}}_i, x \neq x_{\tau,i}} (x>0), a, \overline{\mathcal{X}}_i} (r,i)$ for $\tau = q \xrightarrow{C,a,X} r \in \delta$ where $a \in \Sigma$, $\eta(q) = \mathcal{X}_i$, $\eta(r) = \mathcal{X}_j$ with $j \neq i$ (source component gets asleep, case of discrete visible transition between distinct components).

   (e) $(q,i) \xrightarrow{C,\varepsilon,X \cup \overline{\mathcal{X}}_i} (\tau,j,i)$ for $\tau = q \xrightarrow{C',\varepsilon,X} r \in \delta$ with $\eta(q) = \mathcal{X}_j$, $\eta(r) = \mathcal{X}_i$ with $i \neq j$, and

   $$C = C' \wedge (x_{\tau,j} = 0) \wedge \bigwedge_{x \in \overline{\mathcal{X}}_j, x \neq x_{\tau,j}} (x > 0) \wedge \bigwedge_{l \neq j} \bigwedge_{x \in \overline{\mathcal{X}}_l} (x = 0)$$

   (target component in transition τ gets awaken, case of ε-transitions between distinct components).

(f) $(\tau, j, i) \xrightarrow{C,\varepsilon,\{x_{\tau,i}\}} (\tau, i)$ for $\tau = q \xrightarrow{C',\varepsilon,X} r \in \delta$ with $\eta(q) = \mathcal{X}_j$, $\eta(r) = \mathcal{X}_i$ with $i \neq j$, and

$$C = (y_i > 0) \wedge (x_{\tau,j} = 0) \wedge \bigwedge_{x \in \overline{\mathcal{X}}_j, x \neq x_{\tau,j}} (x > 0) \wedge \bigwedge_{l \neq i,j} \bigwedge_{x \in \overline{\mathcal{X}}_l} (x = 0)$$

(target components acknowledges source component in τ that it is awake, case of ε-transitions between distinct components). After such a transition is taken, between all clocks in $\overline{\mathcal{X}}_i$ and $\overline{\mathcal{X}}_j$ only $x_{\tau,i}$ and $x_{\tau,j}$ are zero, property which is "known" both by component i and component j. This distinguishes the configuration reached after this transition from configurations reached from all the other transitions.

(g) $(\tau, i) \xrightarrow{C,\varepsilon,\overline{\mathcal{X}}_i} (r, i)$ for $\tau = q \xrightarrow{C',\varepsilon,X} r \in \delta$ with $\eta(q) = \mathcal{X}_i$, $\eta(r) = \mathcal{X}_j$ with $i \neq j$, and

$$C = (x_{\tau,i} = 0) \wedge (x_{\tau,j} = 0) \wedge \bigwedge_{x \in \overline{\mathcal{X}}_i \cup \overline{\mathcal{X}}_j, x \neq x_{\tau,i}, x \neq x_{\tau,j}} (x > 0) \wedge \bigwedge_{x \in \mathcal{X}_{\mathcal{B}} \setminus (\overline{\mathcal{X}}_i \cup \overline{\mathcal{X}}_j)} (x = 0)$$

(source component receives acknowledgment from target component in τ about its wakeup, then goes asleep, case of ε-transitions between distinct components).

(h) $(\tau, i) \xrightarrow{C,\varepsilon,\emptyset} (r, i)$ for $\tau = q \xrightarrow{C',\varepsilon,X} r \in \delta$ with $\eta(q) = \mathcal{X}_j$, $\eta(r) = \mathcal{X}_i$ with $i \neq j$, and

$$C = (x_{\tau,i} = 0) \wedge \bigwedge_{x \in \overline{\mathcal{X}}_i, x \neq x_{\tau,i}} (x > 0) \wedge \bigwedge_{l \neq i} \bigwedge_{x \in \overline{\mathcal{X}}_l} (x = 0)$$

(target component sees source component got asleep and hence may continue its execution, case of ε-transitions between distinct components).

(i) $(q, i) \xrightarrow{C,\xi,\emptyset} (r, i)$ if $\tau = q \xrightarrow{C',\xi,X} r \in \delta$, $\eta(q) = \mathcal{X}_{j_1}$, $\eta(r) = \mathcal{X}_{j_2}$, with $j \neq i$, $j_2 \neq i$ and

$$C = C' \wedge (x_{\tau,j_1} = 0) \wedge \bigwedge_{x \in \overline{\mathcal{X}}_{j_1}, x \neq x_{\tau,j_1}} (x > 0) \wedge \bigwedge_{x \in \mathcal{X}_{\mathcal{B}} \setminus \overline{\mathcal{X}}_{j_1}} (x = 0) \wedge \bigwedge_{x \in X} (x = 0)$$

(inactive components update their local state according to their guess of the current transition, based on their observations on clocks that have just been reset).

(j) $(q, i) \xrightarrow{C,\varepsilon,\{x_{\tau,i}, x_{\overline{\tau},i}\}} (\tau, \overline{\tau}, i)$ if $\tau = q \xrightarrow{C',a,X} r \in \delta$, $\overline{\tau} = \overline{q} \xrightarrow{\overline{C},\overline{a},\overline{X}} \overline{r} \in \delta$, $a, \overline{a} \in \Sigma$, $\eta(q) = \eta(r) = \eta(\overline{q}) = \mathcal{X}_i$, and

$$C = C' \wedge (y_i > 0) \wedge \bigwedge_{x \in \mathcal{X}_{\mathcal{B}} \setminus \overline{\mathcal{X}}_i} (x = 0)$$

(source component broadcast its intention to execute *in zero time* a sequence of discrete visible transitions which starts with τ and ends with τ').

(k) $(\tau, \overline{\tau}, i) \xrightarrow{C,a,X} (r, \overline{\tau}, i)$ if $\tau = q \xrightarrow{C',a,X} r$, $\overline{\tau} = \overline{q} \xrightarrow{\overline{C},\overline{a},\overline{X}} \overline{r} \in \delta$, $a, \overline{a} \in \Sigma$, $\eta(q) = \eta(r) = \eta(\overline{q}) = \mathcal{X}_i$ and

$$C = C' \wedge (x_{\tau,i} = 0) \wedge (x_{\overline{\tau},i} = 0) \wedge \bigwedge_{x \in \overline{\mathcal{X}}_i, x \neq x_{\tau,i}, x \neq x_{\overline{\tau},i}} (x > 0) \wedge \bigwedge_{x \in \mathcal{X}_{\mathcal{B}} \setminus \overline{\mathcal{X}}_i} (x = 0)$$

(source component executes the first transition in the sequence of discrete visible transitions that it wants to execute in zero time).

(l) $(q_1, \overline{\tau}, i) \xrightarrow{C_1', a_1, X_1} (r_1, \overline{\tau}, i)$ if there exist δ-transitions $\tau = q \xrightarrow{C,a,X} r$, $\overline{\tau} = \overline{q} \xrightarrow{\overline{C},\overline{a},\overline{X}} \overline{r}$, $q_1 \xrightarrow{C_1,a_1,X_1} r_1$, with $a, a_1, \overline{a} \in \Sigma$, $\eta(q) = \eta(q_1) = \eta(r) = \eta(r_1) = \eta(\overline{q}) = \mathcal{X}_i$ and

$$C_1' = C_1 \wedge (x_{\tau,i} = 0) \wedge (x_{\overline{\tau},i} = 0) \wedge \bigwedge_{x \in \overline{\mathcal{X}}_i, x \neq x_{\tau,i}, x \neq x_{\overline{\tau},i}} (x > 0) \wedge \bigwedge_{x \in \mathcal{X}_{\mathcal{B}} \setminus \overline{\mathcal{X}}_i} (x = 0)$$

(source component continues to execute discrete visible transitions in zero time).

(m) $(\overline{q}, \overline{\tau}, i) \xrightarrow{C,\overline{a},\overline{X} \cup \overline{\mathcal{X}}_i} (\overline{r}, \overline{\tau}, i)$ if there exist δ-transitions $\tau = q \xrightarrow{C',a,X} r$, $\overline{\tau} = \overline{q} \xrightarrow{\overline{C},\overline{a},\overline{X}} \overline{r}$ with $a, \overline{a} \in \Sigma$, $q', r' \in Q$, $\eta(q) = \eta(r) = \eta(\overline{q}) = \eta(\overline{r}) = \mathcal{X}_i$ and

$$C = \overline{C} \wedge (x_{\tau,i} = 0) \wedge (x_{\overline{\tau},i} = 0) \wedge \bigwedge_{x \in \overline{\mathcal{X}}_i, x \neq x_{\tau,i}, x \neq x_{\overline{\tau},i}} (x > 0) \wedge \bigwedge_{x \in \mathcal{X}_{\mathcal{B}} \setminus \overline{\mathcal{X}}_i} (x = 0)$$

(source component ends its sequence of visible transitions taken in zero time, target component is the same as the source component).

(n) $(\overline{q}, \overline{\tau}, i) \xrightarrow{C,\overline{a},\emptyset} (\overline{q}, \overline{r}, \overline{\tau}, i)$ if there exist δ-transitions $\tau = q \xrightarrow{C',a,X} r$, $\overline{\tau} = \overline{q} \xrightarrow{\overline{C},\overline{a},\overline{X}} \overline{r}$, with $a, \overline{a} \in \Sigma$, $\eta(q) = \eta(r) = \eta(\overline{q}) = \mathcal{X}_i$, $\eta(\overline{r}) = \mathcal{X}_j$ with $j \neq i$ and

$$C = \overline{C} \wedge (x_{\tau,i} = 0) \wedge (x_{\overline{\tau},i} = 0) \wedge \bigwedge_{x \in \overline{\mathcal{X}}_i, x \neq x_{\tau,i}, x \neq x_{\overline{\tau},i}} (x > 0) \wedge \bigwedge_{x \in \mathcal{X}_{\mathcal{B}} \setminus \overline{\mathcal{X}}_i} (x = 0)$$

(source component ends its sequence of visible transitions taken in zero time, waits to be informed that target component j awoke itself).

(o) $(q, i) \xrightarrow{C,a,\emptyset} (r, \overline{\tau}, i)$ if there exist δ-transitions $\tau = q \xrightarrow{C',a,X} r$, $\overline{\tau} = \overline{q} \xrightarrow{\overline{C},\overline{a},\overline{X}} \overline{r}$, with $a, \overline{a} \in \Sigma$, $\eta(q) = \eta(r) = \eta(\overline{q}) = \mathcal{X}_j$, with $j \neq i$ and

$$C = C' \wedge (x_{\tau,j} = 0) \wedge (x_{\overline{\tau},j} = 0) \wedge \bigwedge_{x \in \overline{\mathcal{X}}_j, x \neq x_{\tau,j}, x \neq x_{\overline{\tau},j}} (x > 0) \wedge \bigwedge_{x \in \mathcal{X}_{\mathcal{B}} \setminus \overline{\mathcal{X}}_j} (x = 0)$$

(inactive components get informed that the active component wants to do a sequence of discrete visible transitions in zero time).

(p) $(q_1, \overline{\tau}, i) \xrightarrow{C_1',a_1,\emptyset} (r_1, \overline{\tau}, i)$ if there exist δ-transitions $\tau = q \xrightarrow{C',a,X} r$, $\overline{\tau} = \overline{q} \xrightarrow{\overline{C},\overline{a},\overline{X}} \overline{r}$, with $\tau_1 = q_1 \xrightarrow{C',a',X'} r_1 \in \delta$, $a, \overline{a} \in \Sigma$, $\eta(q) = \eta(q_1) = \eta(r) = \eta(r_1) = \eta(\overline{q}) = \mathcal{X}_j$, with $j \neq i$ and

$$C_1' = C_1 \wedge (x_{\tau,j} = 0) \wedge (x_{\overline{\tau},j} = 0) \wedge \bigwedge_{x \in \overline{\mathcal{X}}_j, x \neq x_{\tau,j}, x \neq x_{\overline{\tau},j}} (x > 0) \wedge \bigwedge_{x \in \mathcal{X}_{\mathcal{B}} \setminus \overline{\mathcal{X}}_j} (x = 0)$$

(inactive components guess the transition that the active component is executing in its sequence of discrete visible transitions taken in zero time).

(q) $(q_1, \overline{\tau}, i) \xrightarrow{C_1', a_1, \emptyset} (r_1, i)$ if there exist δ-transitions $\tau = q \xrightarrow{C', a, X} r$, $\overline{\tau} = \overline{q} \xrightarrow{\overline{C}, \overline{a}, \overline{X}} \overline{r}$, with $\tau_1 = q_1 \xrightarrow{C', a', X'} r_1 \in \delta$, $a, \overline{a} \in \Sigma$, $\eta(q) = \eta(q_1) = \eta(r) = \eta(r_1) = \eta(\overline{q}) = \mathcal{X}_j$, with $j \neq i$ and

$$C_1' = C_1 \wedge (x_{\tau,j} = 0) \wedge (x_{\overline{\tau},j} = 0) \wedge \bigwedge_{x \in \overline{\mathcal{X}}_j, x \neq x_{\tau,j}, x \neq x_{\overline{\tau},j}} (x > 0) \wedge \bigwedge_{x \in \mathcal{X}_{\mathcal{B}} \setminus \overline{\mathcal{X}}_j} (x = 0)$$

(inactive components guess that the active component ends its sequence of discrete visible transitions taken in zero time).

(r) $(\overline{q}, \overline{\tau}, i) \xrightarrow{C, \overline{a}, \overline{X}} (\overline{q}, \overline{r}, \overline{\tau}, i)$ if there exist δ-transitions $\tau = q \xrightarrow{C', a, X} r$, $\overline{\tau} = \overline{q} \xrightarrow{\overline{C}, \overline{a}, \overline{X}} \overline{r}$, with $\eta(q) = \eta(r) = \eta(\overline{q}) = \mathcal{X}_j$, $\eta(\overline{r}) = \mathcal{X}_i$ with $j \neq i$ and

$$C = \overline{C} \wedge (x_{\tau,j} = 0) \wedge (x_{\overline{\tau},j} = 0) \wedge \bigwedge_{x \in \overline{\mathcal{X}}_j, x \neq x_{\tau,j}, x \neq x_{\overline{\tau},j}} (x > 0) \wedge \bigwedge_{x \in \mathcal{X}_{\mathcal{B}} \setminus \overline{\mathcal{X}}_i} (x = 0)$$

(target component in transition $\overline{\tau}$ awakes itself as it guesses that source component executes $\overline{\tau}$ and ends the sequence of visible transitions taken in zero time).

(s) $(\overline{q}, \overline{r}, \overline{\tau}, i) \xrightarrow{y_i > 0, \varepsilon, \{x_{\overline{\tau},i}\}} (\overline{r}, \overline{\tau}, i)$ if there exist δ-transitions $\tau = q \xrightarrow{C', a, X} r$, $\overline{\tau} = \overline{q} \xrightarrow{\overline{C}, \overline{a}, \overline{X}} \overline{r}$, with $\eta(q) = \eta(r) = \eta(\overline{q}) = \mathcal{X}_j$, $\eta(\overline{r}) = \mathcal{X}_i$ with $j \neq i$. and

$$C = (y_i > 0) \wedge (x_{\tau,j} = 0) \wedge (x_{\overline{\tau},j} = 0) \wedge \bigwedge_{x \in \overline{\mathcal{X}}_j, x \neq y_j, x \neq x_{\overline{\tau},j}} (x > 0) \wedge \bigwedge_{x \in \mathcal{X}_{\mathcal{B}} \setminus (\overline{\mathcal{X}}_i \cup \overline{\mathcal{X}}_j)} (x = 0)$$

(target component acknowledges source component that it it is awake after the end of the sequence of discrete visible transitions taken in zero time).

(t) $(\overline{q}, \overline{r}, \overline{\tau}, i) \xrightarrow{C, \overline{a}, \overline{\mathcal{X}}_i} (\overline{r}, i)$ if there exist δ-transitions $\tau = q \xrightarrow{C', a, X} r$, $\overline{\tau} = \overline{q} \xrightarrow{\overline{C}, \overline{a}, \overline{X}} \overline{r}$, with $\eta(q) = \eta(r) = \eta(\overline{q}) = \mathcal{X}_i$, $\eta(\overline{r}) = \mathcal{X}_j$ with $j \neq i$ and

$$C = (y_j > 0) \wedge (x_{\overline{\tau},j} = 0) \wedge (x_{\tau,i} = 0) \wedge (x_{\overline{\tau},i} = 0) \wedge \bigwedge_{x \in \overline{\mathcal{X}}_j, x \neq y_j, x \neq x_{\overline{\tau},j}} (x > 0) \wedge \bigwedge_{x \in \mathcal{X}_{\mathcal{B}} \setminus (\overline{\mathcal{X}}_i \cup \overline{\mathcal{X}}_j)} (x = 0)$$

(source component gets informed that target component is awake at the end of the sequence of discrete visible transitions taken in zero time, hence it may go asleep).

(u) $(\overline{r}, \overline{\tau}, i) \xrightarrow{C, \overline{a}, \emptyset} (\overline{r}, i)$ if there exist δ-transitions $\tau = q \xrightarrow{C', a, X} r$, $\overline{\tau} = \overline{q} \xrightarrow{\overline{C}, \overline{a}, \overline{X}} \overline{r}$, with $\eta(q) = \eta(r) = \eta(\overline{q}) = \mathcal{X}_j$, $\eta(\overline{r}) = \mathcal{X}_i$ with $j \neq i$ and

$$C = (x_{\overline{\tau},i} = 0) \wedge \bigwedge_{x \in \mathcal{X}_{\mathcal{B}} \setminus \overline{\mathcal{X}}_i} (x = 0)$$

(target component sees source component got asleep after a sequence of discrete visible transitions executed in zero time and whose last transition is $\overline{\tau}$, and hence target component may continue its execution).

(v) $(q,0,i) \xrightarrow{C,a,X} (r,0,i)$ and $(q,0,i) \xrightarrow{C,a,X} (r,i)$ if $q \xrightarrow{C',a,X} r \in \delta$ with

$$C = C' \wedge \bigwedge_{x \in \mathcal{X}_{\mathcal{B}}} (x = 0)$$

These transitions can be taken at the beginning of each trajectory, if the trajectory starts with a sequence of discrete transitions taken in zero time.

The inclusion $L(\mathcal{A}) \supseteq L(\mathcal{B})$ is straightforward as each trajectory in $\mathcal{A}$ can be simulated in $\mathcal{B}$.

For the reverse inclusion, take a run in $\mathcal{B}$, $\overline{\rho} = (\rho_i)_{1 \leq i \leq n}$, with $\rho_i = (q^i_{j-1} \xrightarrow{C^i_j, \xi_j, X^i_j} q^i_j)_{1 \leq i \leq m}$, and a trajectory $trj = \big((q^1_{j-1}, \ldots, q^n_{j-1}, v_{j-1}) \xrightarrow{\overline{\xi}_j} (q^1_j, \ldots, q^n_j, v_j)\big)_{1 \leq j \leq 2m}$, with $\overline{\rho}$ associated with trj.

We say that component i is **sleeping during time passage interval** $2j-1$ if $\xi_{2j-1} > 0$ and $v_{2j-2}\big|_{\tilde{\mathcal{X}}_i} = v_{2j-1}\big|_{\tilde{\mathcal{X}}_i}$. We also say that some component i is **awake during time passage interval** $2j-1$ if $v_{2j-2}\big|_{\tilde{\mathcal{X}}_i} \neq v_{2j-1}\big|_{\tilde{\mathcal{X}}_i}$.

A third type of situations might occur in some time intervals in trj: the time intervals for which $\xi_{2j-1} = 0$. Within such intervals, we consider that the component that is awake is the last component that was awake during a previous non-zero time interval. Finally, trj might start with a sequence of transitions taken in zero time, and therefore there is no component that is awake before this sequence of transitions. We consider by convention that, during such sequences of transitions, the only component that is active is the component whose set of stopwatches is the same as $\eta(q_0)$, where q_0 is the distinguished initial state in the definition of state-region determinism.

We denote $awake(2j-1) = i$ if during time passage interval $2j-1$ component i is awake. By extension, we also denote $awake(2j) = i$ in this case.

We may then prove, by induction on the indices j of the transitions in the trajectory trj, the following properties

1. The definition of $awake(j)$ is correct, that is, for each $1 \leq j \leq 2m$ there exists exactly one component i which is active during time passage interval j, if j is odd, resp $j-1$ otherwise.
2. If, at time interval 1, the active component is i and starts in state (q,i) with $q \in Q_0$, $q \neq q_0$, then it must pass a non-zero amount of time in (q,i) and therefore when it decides to take an action, by means of transitions of type (a), any component that has incorrectly guessed the initial state will block the whole distributed time-asynchronous automaton. On the other hand, if the active component starts in state (q_0,i) (where $q_0 \in Q_0$ is the distinguished state corresponding to sequences of transitions taken in zero time), then all components must start in q_0, or otherwise again the whole trajectory will be blocked by those components which made the incorrect guess of the initial state.
3. If component i is active during time passage interval $2j_1 - 1$ and during time passage interval $2j_2 - 1$ with $j_2 > j_1$, no component is active during any time passage intervals between $2j_1 - 1$ and $2j_2 - 1$, there exists a unique visible discrete transition in trj indexed between $2j_1$ and $2j_2 - 2$ and no transition of type (e) or (j) is indexed between $2j_1$ and $2j_2 - 2$, then between these two moments component i executes a transition of type (a) followed by a transition

of type (b), and all the other components execute a transition of type (i). Moreover, at the end of these transitions, all components correctly guess the state of the active component, i.e. $q^i_{2j_2-1} = q^l_{2j_2-1}$ for all l.

4. If component i_1 is active during time passage interval $2j_1 - 1$, component i_2 is active during time passage interval $2j_2-1$, with $i_1 \neq i_2$ and $j_2 > j_1$, no component is active during any time passage intervals between $2j_1-1$ and $2j_2-1$, there exists a unique visible discrete transition in trj indexed between $2j_1 - 1$ and $2j_2 - 1$ and no transition of type (e) or (j) is indexed between $2j_1$ and $2j_2 - 2$, then between these two moments component i_1 executes a transition of type (a) followed by a transition of type (d), while component i_2 executes a transition of type (c). and all the other components execute a transition of type (i). Moreover, at the end of these transitions, all components agree on the current state, i.e. $q^{i_1}_{2j_2-1} = q^{i_2}_{2j_2-1} = q^l_{2j_2-1}$ for all l.
5. If component i_1 is active during time passage interval $2j_1 - 1$, component i_2 is active during time passage interval $2j_2 - 2$, with $i_1 \neq i_2$ and $j_2 > j_1$, no component is active during any time passage intervals between $2j_1 - 1$ and $2j_2 - 2$, and there exists no discrete transition indexed between $2j_1$ and $2j_2 - 2$ which is of type (b), (c), (d) or (j), then between these two moments component i_1 executes a transition of type (a) followed by a transition of type (g), while component i_2 executes a transition of type (e) followed by a transition of type (f) and by a third transition of type (h). Moreover, at the end of these transitions, components i_1 and i_2 agree on the current state, i.e. $q^{i_1}_{2j_2-1} = q^{i_2}_{2j_2-1}$.
   Note that, in this case, it might happen that not all the other components execute a transition of type (i).
6. If component i is active during time passage interval $2j_1 - 1$ and during time passage interval $2j_2 - 1$, with $j_2 > j_1$, no component is active during any time passage intervals between $2j_1$ and $2j_2 - 1$, and all the discrete and visible transitions $2j$ with $j_1 \leq j < j_2$ are separated by zero delays, i.e. $\xi_{2j+1} = 0$, then between these two moments component i executes a transition of type (j) followed by transition of type (k) and a succession of transitions of type (l) and finished by a transition of type (m), and all the other components execute transitions of type (o), (p), resp. (q). Moreover, at the end of these transitions, all components agree on the current state, i.e. $q^i_{2j_2-1} = q^l_{2j_2-1}$ for all l.
7. If component i_1 is active during time passage interval $2j_1 - 1$, component i_2 is active during time passage interval $2j_2 - 1$, with $i_1 \neq i_2$ and $j_2 > j_1$, no component is active during any time passage intervals between $2j_1$ and $2j_2 - 2$, and all the discrete and visible transitions $2j$ with $j_1 \leq j < j_2$ are separated by zero delays, i.e. $\xi_{2j+1} = 0$, then between these two moments component i_1 executes a transition of type (j) followed by transition of type (k) and a succession of transitions of type (l) and finished by a transition of type (n) and then a transition of type (t), component i_2 executes a succession of transitions of type (o) followed by a sequence of transitions of alternating types (p) and (q) and finished by a transition of type (r), then a transition of type (s) and a transition of type (u), and all the other components execute transitions of type (o), (p), resp. (q). Moreover, at the end of these transitions, components i_1 and i_2 agree on the current state, i.e. $q^{i_1}_{2j_2-1} = q^{i_2}_{2j_2-1}$.
8. Any sequence of discrete transitions that are taken in zero time are executed by a unique component, which is the component corresponding to the unique initial state q_0 in Definition 4, the only initial state from where such sequences of discrete transitions can be started in $\mathcal{A}$.

9. The following sequence

$$trj' = \left((q_{j-1}^{awake(j-1)}, v_{j-1}\big|_{\mathcal{X}}) \xrightarrow{\overline{\xi}_j} (q_j^{awake(j)}, v_j\big|_{\mathcal{X}})\right)_{1\leq j\leq 2m}$$

is a trajectory in $\mathcal{A}$, and if trj is accepting then trj' is accepting too.

This ends the proof of Theorem 9. □

4.1 Non-closure of partitioned stopwatch automata under intersection

This section contains the proof of the non-closure result from Proposition 3:

Proposition 5. *The class of languages accepted by distributed time-asynchronous automata – and hence the class of partitioned stopwatch automata – are not closed under intersection.*

Let us first give the counterexample that will be used in the sequel for proving the non-closure property:

$$L = \left\{t_1 a t_2 a t_3 a t_4 a \mid t_1, t_2, t_3, t_4 \in]0,1[\,, t_1 + t_3 = t_2 + t_4 = t_1 + t_4 = 1\right\}$$

L is the intersection of the language of the distributed time-asynchronous automaton in Figure 7 with the language of the distributed time-asynchronous automaton in Figure 8, and is the language accepted by the stopwatch automaton of Figure 2.

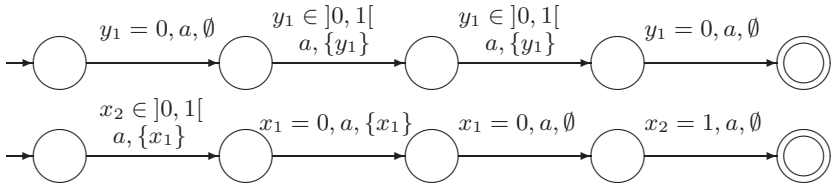

Fig. 8. The two components of a distributed time-asynchronous automaton recognizing the language $\{t_1\, a\, t_2\, a\, t_3\, a\, t_4\, a \mid t_1, t_2, t_3, t_4 \in]0,1[\,, t_1 + t_4 = 1\}$

Intuitively, the constraint defining L cannot be simulated by an asynchronous composition of timed automata, as it needs some stopwatch automaton in which the distribution of stopwatches is not a partition. In particular, the stopwatch that checks the constraint $t_1 + t_3 = 1$ and the stopwatch that checks the constraint $t_1 + t_4$ must be simultaneously active in the location where the time passage of t_1 happens, and exactly one of them must be active in the locations where the time passage of t_2 or t_3 happen.

The balance of this section gives the formalization of the above intuitions. We will start by introducing a subclass of distributed time-asynchronous automata in which components can only read their own clocks, and not the clocks owned by other components, and then proving a series of useful technical properties.

Formally, a **distributed time-asynchronous automaton with private clocksets** is a distributed time-asynchronous automaton in which each component i can only test clocks in $\mathcal{X}_i$, i.e., for all $1 \leq i \leq n,$ if $q \xrightarrow{C,a,X} q' \in \delta_i$ then $C \in \mathsf{Constr}(\mathcal{X}_i)$.

The following property shows the connection between distributed time-asynchronous automaton and distributed time-asynchronous automaton with private clocksets:

Proposition 6. *For each distributed time-asynchronous automaton $\mathcal{A}$ with set of symbols Σ there exists a distributed time-asynchronous automaton with private clocksets and with some set of symbols $\overline{\Sigma}$ and a renaming $f : \overline{\Sigma} \to \Sigma \cup \{\varepsilon\}$ such that $L(\mathcal{A}) = f\big(L(\mathcal{B})\big)$.*

Moreover, if $\mathcal{A}$ contains no ε-transition, then f contains no symbol deletion.

Proof. The proof idea is the same as in the proof of the Kleene Theorem 5: we use a new set of symbols that will be used to label distinct transitions with distinct symbols. The new symbols will represent tuples of transitions, one in each component.

So start with $\mathcal{A} = (Q_1, \ldots, Q_n, \Sigma, \mathcal{X}_1, \ldots, \mathcal{X}_n, \delta_1, \ldots, \delta_n, Q_0^1, \ldots, Q_0^n, Q_f^1, \ldots Q_f^n)$ and $\mathcal{B} = (\overline{Q}_1, \ldots, \overline{Q}_n, \overline{\Sigma}, \mathcal{X}_1, \ldots, \mathcal{X}_n, \overline{\delta}_1, \ldots, \overline{\delta}_n, \overline{Q}_0^1, \ldots, \overline{Q}_0^n, \overline{Q}_f^1, \ldots \overline{Q}_f^n)$. If we recall that $Q = Q_1 \times \ldots \times Q_n$, $Q_f = Q_f^1 \times \ldots \times Q_f^n$ and $Q_0 = Q_0^1 \times \ldots \times Q_0^n$, then, formally, the components of $\mathcal{B}$ are:

- The set of locations is $\overline{Q}_i = Q \times \{i\}$, with $\overline{Q}_0^i = Q_0 \times \{i\}$ and $Q_f^i = Q_f \times \{i\}$.
- The set of input symbols is

$$\overline{\Sigma} = \big\{(tr_1, \ldots, tr_n) \mid \text{ for all } 1 \leq i \leq n, tr_i = q \xrightarrow{C,\xi,Y} r \in \delta_i\big\}$$

- The i-th transition relation is:

$$\overline{\delta}_i = \big\{(\overline{q}, i) \xrightarrow{\overline{C}_i,\xi,Y_i} (\overline{r}, i) \mid \overline{q} = (q_1, \ldots, q_n), \overline{r} = (r_1, \ldots, r_n),$$
$$\text{for all } j \leq n \text{ there exists } tr_j = q_j \xrightarrow{C_j,b,Y_j} r_j \in \delta_j \text{ with } \xi = (tr_1, \ldots, tr_n), \overline{C}_i = C_i\big\vert_{\mathcal{X}_i}\big\}\big\}$$

An easy but tedious double inclusion proves that $L(\mathcal{A}) = f(L(\mathcal{B}))$. □

In the sequel, for each $1 \leq i \leq n$, denote $\mathcal{A}_i$ the timed automaton corresponding to the i-th component of $\mathcal{A}$:

$$\mathcal{A}_i = (Q_i, \Sigma, \mathcal{X}_i, \delta_i, Q_0^i, Q_f^i)$$

We will now show that the language of a distributed time-asynchronous automaton with private clocksets is, in some sense, a "synchronized product" of the languages of $\mathcal{A}_i$. The exact meaning of the term "synchronization" is the following:

Definition 8. *Given n timed words $w_i = t_1^i a_1 \ldots t_k^i a_k t_{k+1}^i$ ($1 \leq i \leq n$), all having the same untiming sequence $a_1, \ldots, a_k$, the **synchronization** of $(w_i)_{1 \leq i \leq n}$, denoted $w_1 \odot \ldots \odot w_n$, is the timed word:*

$$w = \Big(\sum_{i=1}^{n} t_1^i\Big) a_1 \ldots \Big(\sum_{i=1}^{n} t_k^i\Big) a_k \Big(\sum_{i=1}^{n} t_{k+1}^i\Big)$$

*Given a distributed time-asynchronous automaton $\mathcal{A}$, a run $\rho = (\overline{q}_{j-1} \xrightarrow{C_j,\xi_j,X_j} \overline{q}_j)_{1 \leq j \leq k}$ in $\mathcal{A}$ and, for each $1 \leq i \leq n$, a run $\rho_i = (q_{j-1}^i \xrightarrow{C_j^i,\xi_j^i,X_j^i} q_j^i)_{1 \leq j \leq k}$ in the component $\mathcal{A}_i$, we say that ρ is an $\mathcal{A}$-**synchronization** of $(\rho_i)_{1 \leq i \leq n}$ if*

For each $1 \leq j \leq k$, $\xi_j = \xi^i_j$, $\bigwedge_{1\leq i\leq n} C^i_j \to C_j$ *is a valid constraint and* $\bigcup_{1\leq i\leq n} X^i_j = X_j$.

Proposition 7. *1. In the setting of the above definition, any $\mathcal{A}$-synchronization of runs $(\rho_j)_{1\leq j\leq k}$ is a run in $\mathcal{A}$, and if each ρ_j is accepting in $\mathcal{A}_j$ then ρ is accepting in $\mathcal{A}$.*
2. For any tuple of timed words w_j ($1 \leq j \leq n$), if each w_j is associated with the run ρ_j, then their synchronization $w_1 \odot \ldots \odot w_j$ is associated with any $\mathcal{A}$-synchronization of $(\rho_j)_{1\leq j\leq k}$.
3. $L(\mathcal{A}) = L(\mathcal{A}_1) \odot \ldots \odot L(\mathcal{A}_n)$.

The proofs of these results are straightforward from the definitions.

The following technical property states that, if a run of a distributed time-asynchronous automaton with private clockset in which the time passage between two symbols must be in the interval $]0, 1[$, then exactly one component may be "active" in between the two symbols (meaning that all the other may only allow a zero time passage). The intuition comes from the fact that the constraints use integers, and we cannot decompose the interval $]0, 1[$ as the sum of two non-zero intervals with integer bounds. (Here, summation is considered in the sense of the following example: $[0, 1[+]1, 2] =]1, 3[$.)

Proposition 8. *Consider a distributed time-asynchronous automaton with private clocksets $\mathcal{A}$ with n components and no ε-transitions, and a run ρ in $\mathcal{A}$, $\rho = \left(\overline{q}_{i-1} \xrightarrow{C_i,a_i,X_i} \overline{q}_i\right)_{1\leq i\leq k}$, with $\overline{q}_i = (q^1_i, \ldots q^n_i)$. Denote ρ_j the projection of ρ in component j, that is, $\rho_j = \left(q^j_{i-1} \xrightarrow{C_i\cap \mathsf{Constr}(\mathcal{X}_j),a_i,X_i\cap\mathcal{X}_j} q^j_i\right)_{1\leq i\leq k}$.*

1. Suppose that there exists some i_0 ($1 \leq i_0 \leq k$) such that for any timed word $w = t_1a_1 \ldots t_ka_k$ associated with ρ, we have $t_{i_0} \in]0, 1[$. Then there exists a unique j_0 such that for any synchronization $w_1 \odot \ldots \odot w_n = w$ in which $w_j = t^j_1a_1 \ldots t^j_ka_k$ is associated with ρ_j, we have that $t^{j_0}_{i_0} \in]0, 1[$ and for all $j \neq j_0$, $t^j_{i_0} = 0$.
2. More generally, if there exist two integers $i_1 < i_2$ such that for any timed word $w = t_1a_1 \ldots t_ka_k$ associated with ρ, we have that $\sum_{i_1\leq i\leq i_2} t_i \in]0, 1[$, then there exists a unique j_0 such that for any synchronization $w_1 \odot \ldots \odot w_n = w$ in which $w_j = t^j_1a_1 \ldots t^j_ka_k$ is associated with ρ_j, we have that $\sum_{i_1\leq i\leq i_2} t^{j_0}_i \in]0, 1[$ and for all $j \neq j_0$, $\sum_{i_1\leq i\leq i_2} t^j_i = 0$.

In other words, the first property states that, if $\mathcal{A}$ accepts only timed words $w = t_1a_1 \ldots t_ka_k$ with $t_i \in]0, 1[$ for some fixed i, then in the run associated with w, only one component may have non-zero time passage between a_{i-1} and a_i, all the other components must have zero time passage. The second property is a generalization of this.

Proof (of Proposition 8). The proof follows from a combination of Remark 4, applied to the runs ρ_j, and of Proposition 7 on synchronization of timed words associated to ρ_j. We will prove only the first property, the second being provable similarly.

So assume, for the sake of contradiction, that there exist two distinct components j_1, j_2 such that ρ_{j_1} is associated with $w_{j_1} = t^{j_1}_1a_1 \ldots t^{j_1}_ka_k$ with, $t^{j_1}_{i_0} \in]0, 1[$, and ρ_{j_2} is associated with $w_{j_2} = t^{j_2}_1a_1 \ldots t^{j_2}_ka_k$ with, $t^{j_2}_{i_0} \in]0, 1[$, where $1 \leq i_0 \leq k$.

But then, by means of Remark 4, we may take any positive real $\alpha \in]0, 1[$ and construct a new timed word $w'_{j_1} = u^{j_1}_1a_1 \ldots u^{j_1}_ka_k$ which is associated with ρ_{j_1} and with $u^{j_1}_{i_0} = \alpha$. The same will hold for ρ_{j_2}.

This gives us the possibility to choose w'_{j_1} such that $u_{i_0}^{j_1} = 0.9$ and w'_{j_2} such that $u_{i_0}^{j_2} = 0.9$. Following Proposition 7, the synchronization $w_1 \odot \ldots \odot w'_{j_1} \odot \ldots \odot w'_{j_2} \odot \ldots \odot w_n$ would still be associated with ρ.

But this synchronization would have as its i_0 time interval a value larger than $t_{i_0}^{j_1} + t_{i_0}^{j_2} = 1.8$, which is in contradiction with the hypothesis, which says that this time interval should always be less than 1. □

Proposition 9 (Nonclosure under intersection for distributed time-asynchronous automata). *The following language does not belong to the class of timed languages accepted by distributed time-asynchronous automata:*

$$L = \left\{ t_1 a t_2 a t_3 a t_4 a \mid t_1, t_2, t_3, t_4 \in]0,1[\,, t_1 + t_3 = t_2 + t_4 = t_1 + t_4 = 1 \right\}$$

Proof. Suppose, for the sake of contradiction, that the language is accepted by a distributed time-asynchronous automaton $\mathcal{A}$. Following Proposition 6, it would then be in the renaming of the language of a distributed time-asynchronous automaton with private clocksets $\mathcal{B}$. Denote further $\mathcal{B}_j$ the j-th component of the automaton $\mathcal{B}$, which is itself a timed automaton (as already noted in the statement of Proposition 7).

So take the timed word $0.3a0.3a0.7a0.7a \in L(\mathcal{A})$. It would be the renaming of a timed word $w = u_1 a_1 \ldots u_k a_k$ accepted by $\mathcal{B}$, and for which, for some $1 \leq i_1 < i_2 < i_3 \leq k$,

$$\sum_{i=1}^{i_1} u_i = \sum_{i=i_1+1}^{i_2} u_i = 0.3 \text{ and } \sum_{i=i_2+1}^{i_3} u_i = \sum_{i=i_3+1}^{k} u_i = 0.7$$

Take further a run ρ that is associated with w, and denote ρ_j ($1 \leq j \leq n$) the projection of ρ in $\mathcal{B}_j$.

Note that we are in the setting of the Proposition 8, since any timed word accepted by ρ would have to have $\sum_{i=1}^{i_1} u_i \in]0,1[$, $\sum_{i=i_1+1}^{i_2} u_i \in]0,1[$, $\sum_{i=i_2+1}^{i_3} u_i \in]0,1[$, resp. $\sum_{i=i_3+1}^{i_4} u_i \in]0,1[$. Therefore, for any synchronization $w_1 \odot \ldots \odot w_n = w$ with $w_j = u_1^j a_1 \ldots u_k^j a_k$ associated with ρ_j, we must have some unique component j_1 for which $\sum_{i=0}^{i_1} u_i^{j_1} \in]0,1[$. The same must hold for i_2, i_3 and i_4 too, so there must exist a unique j_2 with $\sum_{i=i_1+1}^{i_2} u_i^{j_2} \in]0,1[$, a unique j_3 with $\sum_{i=i_2+1}^{i_3} u_i^{j_3} \in]0,1[$ and a unique j_4 with $\sum_{i=i_3+1}^{k} u_i^{j_4} \in]0,1[$.

Our first task is to prove $j_1 = j_2 = j_3 = j_4$. We prove this by contradiction: assume $j_1 \neq j_3$. By Remark 4, we may construct a timed word with $w'_{j_1} = v_1^{j_1} a_1 \ldots v_k^{j_1} a_k$ associated with ρ_{j_1}, in which $\sum_{i=0}^{i_1} v_i^{j_1} = 0.9$. Similarly, we may construct a timed word with $w'_{j_3} = v_1^{j_3} a_1 \ldots v_k^{j_3} a_k$ associated with ρ_{j_3}, in which $\sum_{i=i_2+1}^{i_3} v_i^{j_3} = 0.9$.

But then clearly the renaming by f of any synchronization comprising w'_{j_1} and w'_{j_3}, which has to be associated with ρ by Proposition 7, would no longer satisfy the constraint $t_1 + t_3 = 1$, since time interval which separates the action a_{j_1} from the action a_{j_3} would have to be at least $\sum_{i=0}^{i_1} v_i^{j_1} + \sum_{i=i_2+1}^{i_3} u_i^{j_3} = 1.8$.

Similarly we may prove that $j_1 = j_4$ and $j_2 = j_4$. But this implies that, in fact, the given distributed time-asynchronous automaton is a timed automaton: by Proposition 8, all the other runs ρ_j would have to be associated to timed words in which all the time intervals are zero.

It is then an easy exercise to repeat the proof of the Proposition 3 to show that our given language L cannot be accepted by a timed automaton. □

5 Modeling timed systems with distributed time-asynchronous automata

In this section we discuss the use of fair shuffle expressions (and hence partitioned stopwatch automata and distributed time-asynchronous automata) to model timed systems.

In particular we consider two examples:

- We show how, given a set of (timed) processes, the behavior of these processes running in a uniprocessor system adopting the Round Robin policy can be modeled with distributed time-asynchronous automata;
- We also show how, given a set of (timed) processes sharing a critical session, the behavior of these processes interacting with a semaphore can be modeled with distributed time-asynchronous automata.

A Round-Robin scheduling model. For the first example, consider $\mathcal{A}_1, \ldots, \mathcal{A}_n$ timed automata expressing n time–dependent processes. We show how to construct a distributed time-asynchronous automaton (equivalently, a fair shuffle expression) expressing the behavior of the processes $\mathcal{A}_1, \ldots, \mathcal{A}_n$ running in a uniprocessor system adopting the Round Robin policy. We suppose that when a process $\mathcal{A}_i$ terminates its computation, it emits a special symbol end_i. Let k be the quantum of time that a process can use the processor continuously. Let k_1 and k_2 be respectively the time necessary to put a process in the waiting queue and to extract a process from the waiting queue. Given a generic process $\mathcal{A}_i$ we modify it in the following manner:

- We introduce a new clock y storing the processor time usage.
- We replace each transition $q \xrightarrow{C,a,X} q'$ with $q \xrightarrow{C \wedge y \leq k,a,X} q'$, in such a manner to give to the process the capability of running if and only if the process has not finished his processor time quantity.
- We add a new state q_{wait} for each state q of $\mathcal{A}_i$ representing that the process is waiting its next turn, after finishing its processor time quantum. We also add two transitions involving q_{wait}:
  - a transition $q \xrightarrow{y=k,stop_i,\{y\}} q_{wait}$ simulating that the process has finished his quantum k and goes in a waiting time.
  - a transition $q_{wait} \xrightarrow{y=0,start_i,\{y\}} q$ simulating that the process woken up.

Let $\mathcal{A}'_1, \ldots, \mathcal{A}'_n$ be the modified timed automata as described above. Consider further n corresponding timed expressions $E_1, \ldots, E_n$, with $\|E_i\| = L(\mathcal{A}'_i)$ for each $1 \leq i \leq n$.

Before defining the Round Robin Scheduler we define one more expression E_{n+1} describing the time $k_1 + k_2$ necessary for switching the context of the enqueued-dequeued processes:

$$E_{n+1} = (\langle enq \ \underline{\mathbf{t}} \ deq \rangle_I)^*, \qquad \text{where } I = [k_1 + k_2, k_1 + k_2]$$

We define next the timed automaton $\mathcal{B} = (Q, \mathcal{X}, \Sigma, \eta, \delta, Q_0, Q_f)$ which represents the Round Robin scheduler:

– Q is the set of triples (i, P, d) where either $i = 0$ and $P = \emptyset$ and $d = extract$, or $i \in \{1, \ldots, n\}$ and $P \subseteq \{1, \ldots, n\}$ and $d \in \{starting, extract, put, running\}$.
The state $(0, \emptyset, extract)$ represents the fact that all the processes have terminated their computations. The states $(i, P, running)$ represents the fact that the running process is $\mathcal{A}_i$ and the non-terminated processes are all $\mathcal{A}_j$ with $j \in P$. The states (i, P, put) (resp. $(i, P, extract)$) represent the fact that the process $\mathcal{A}_i$ is put (resp. is extracted) to (resp. from) the queue. The states $(i, P, starting)$ represent the fact that the process $\mathcal{A}_i$ is about to start its execution.
– $\mathcal{X} = \{x\}$, where x will be a new clock that will be used to ensure that some step of $\mathcal{B}$ is performed instantaneously.
– $\Sigma = \{enq, deq\} \cup \bigcup_{i=1}^{n} \{start_i, stop_i, end_i\}$.
– δ is composed of five types of transitions:
  - $(i, P, running) \xrightarrow{true, stop_i, \{x\}} (i, P, put)$. This transition specifies the fact that the process $\mathcal{A}_i$ has finished his processor time quantity.
  - $(i, P, put) \xrightarrow{x=0, enq, \{x\}} (i, P, extract)$. This transition specifies the fact that the enqueued process for $\mathcal{A}_i$ is started.
  - $(i, P, extract) \xrightarrow{true, deq, \{x\}} (j, P, starting)$ with $j = min(P)$ if $i = max(P)$ and $j = min\{h \in P \mid h > i\}$ otherwise. This transition specifies the fact that the dequeued process for $\mathcal{A}_j$ is finished, where j represents the index of the process that succeeds to i in P.
  - $(i, P, starting) \xrightarrow{x=0, start_i, \{x\}} (i, P, running)$. This transition specifies the fact that the process $\mathcal{A}_j$ is woken up.
  - $(i, P, running) \xrightarrow{true, end_i, \{x\}} (i, P', extract)$ with $P' = P \setminus \{i\}$. This transition specifies that the process $\mathcal{A}_i$ has terminated his computation and hence a new process among the waiting processes in $P \setminus \{i\}$ must be woken up.
– $Q_0 = \{(1, \{1, \ldots, n\}, running)\}$, meaning that, at the beginning, the first processes to run is $\mathcal{A}_1$ and no process has finished its computation.
– $Q_f = \{(0, \emptyset, extract)\}$, specifying that the whole computation terminates when all the processes have terminated their computations.

Let E_{n+2} be the zero-constrained expression equivalent with the timed automaton $\mathcal{B}$. Finally, let f be the renaming which deletes symbols $start_i, stop_i, enq, deq$ (and also end_i if needed). The expression

$$f((E_1 \sqcup \ldots \sqcup E_n \sqcup E_{n+1}) \wedge E_{n+2})$$

is the fair shuffle expression simulating the behavior of the processes $\mathcal{A}_1, \ldots, \mathcal{A}_n$ running in a uniprocessor system adopting the Round Robin policy.

Remark 7. Note however that we do not have global timing constraints, in the sense of *deadlines* for each process. Such constraints would have to be modeled in $\mathcal{B}$, and then require intersection of fair regular expressions with a timed regular expression, hence the result would no longer be a partitioned stopwatch automaton.

A partitioned stopwatch automaton model for semaphores. We show now how to model concurrent timed processes utilizing a semaphore. Consider again n timed automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$ expressing n time-dependent processes. Each process $\mathcal{A}_i$ uses symbols $lock_i$ and $unlock_i$ to operate on the binary semaphore. We suppose that the semaphore implements a LIFO policy.

We can modify each $\mathcal{A}_i$ such that, after each transition labeled $lock_i$, it needs to receive a special symbol ok_i before entering in the critical session. More formally, given a generic process $\mathcal{A}_i$ we modify it by substituting each transition $q \xrightarrow{C,lock_i,X} q'$ with the two transitions $q \xrightarrow{C,lock_i,X} (q', wait)$ and $(q', wait) \xrightarrow{true,ok_i,\emptyset} q'$ where $(q', wait)$ is a new state representing the fact that the process has been enqueued and is waiting the "ok" by the semaphore. Let $\mathcal{A}'_1, \ldots, \mathcal{A}'_n$ be the timed automata modified as described above, and $E_1, \ldots, E_n$, respectively, the timed regular expressions equivalent with each of them.

We then construct a timed automaton $\mathcal{B} = (Q, \mathcal{X}, \Sigma, \eta, \delta, Q_0, Q_f)$ expressing the sequence of accesses to the critical session. Its component are as follows:

- Q is the set of tuples $(d, (i_1, \ldots, i_m))$ where $d \in \{manage, running\}$, and $m \geq 0$, $i_j \in \{1, \ldots, n\}$ and $i_j \neq i_{j'}$ if $j \neq j'$. In a state $(running, (i_1, \ldots, i_m))$, the index i_1 represents the fact that process $\mathcal{A}_{i_1}$ is inside its critical session, and the sequence $i_2, \ldots, i_m$ represents the queue of processes waiting to enter their critical sessions. The state $(manage, (i_1, \ldots, i_m))$ represents that a process has left the critical session by performing an unlock, and hence, a new process among $\mathcal{A}_{i_1}, \ldots, \mathcal{A}_{i_m}$ could enter in the critical session.
- $\mathcal{X} = \emptyset$.
- $\Sigma = \bigcup_{i=1}^{n} \{lock_i, ok_i, unlock_i\}$.
- $\eta((d, (i_1, \ldots, i_m))) = \{x\}$.
- δ is composed of three types of transitions:
  - $(running, (i_1, \ldots, i_m)) \xrightarrow{true,unlock_{i_1},\{\}} (manage, (i_2, \ldots, i_m))$ with $m \geq 1$, meaning that process $\mathcal{A}_{i_1}$ has released the unlock.
  - $(manage, (i_1, \ldots, i_m)) \xrightarrow{true,ok_{i_1},\{\}} (running, (i_1, \ldots, i_m))$ with $m \geq 1$, meaning that the semaphore is not busy and hence the first process $\mathcal{A}_{i_1}$ in the queue is allowed to enter its critical session.
  - $(d, (i_1, \ldots, i_m)) \xrightarrow{true,lock_{i_{m+1}},\{\}} (d, (i_1, i_2, \ldots, i_m i_{m+1}))$ with $d \in \{running, manage\}$, namely the process $\mathcal{A}_{i_{m+1}}$ requires a lock on the critical session and hence is enqueued.
- $Q_0 = \{(manage, (\epsilon))\}$, meaning that, at the beginning, no process has performed a lock.
- $Q_f = \{(manage, (\epsilon))\}$, specifying that there are no more pending locks.

Let E_{n+1} be the zero-constrained expression equivalent with the timed automaton $\mathcal{B}$. Finally, let f be the renaming function that deletes all symbols ok_i (and also $lock_i$ and $unlock_i$ if needed). The expression

$$f((E_1 \sqcup \ldots \sqcup E_n) \wedge E_{n+1})$$

is a fair shuffle expression expressing the behavior of the processes $\mathcal{A}_1, \ldots, \mathcal{A}_n$ running in a uniprocessor system and utilizing a semaphore for entering their critical session.

6 Conclusion

Figure 9 summarizes the main results for partitioned, stopwatch, resp. timed automata and the different types of regular expressions. We mark a property with symbol "*" to represent the fact that the result is proved in this paper.

Model	Closure	Nonclosure	Decidability of reachability
stopwatch automata $=^*$ timed shuffle expressions	union*, intersection*, renaming*, Kleene star*	–	No
$\supsetneq^*$			
partitioned stopwatch automata $=^*$ fair shuffle expressions $=^*$ distributed time-asynchronous automata	union*, renaming*, Kleene star*	intersection*	Yes*
$\supsetneq^*$			
timed automata = timed regular expressions	union, intersection, renaming, Kleene star	–	Yes

Fig. 9. Summary of the main results

The equivalence between partitioned stopwatch automata and distributed time-asynchronous automata presented here is more involved that in our previous paper [DL07], due to the use of clock resets to zero, and not to generalized clock resets as in [DL07]. The communication mechanism between the components of the distributed time-asynchronous automaton that distributes the centralized control of a to-be-simulated partitioned stopwatch automaton works by ensuring that, during each "handshake" between what we called "source component" and "target component" of a transition, both components achieve a form of *common knowledge* ensuring them that the other is consistently simulating the behavior of the partitioned stopwatch automaton. It would be interesting to investigate whether general real-time epistemic frameworks like in [Dim09,WL05] can be used to get a simpler proof of our simulation result.

Another interesting point to be studied is the relationship between partitioned stopwatch automata and the class of Interrupt Timed Automata studied in [BH09], especially related with the fact that the latter, though utilizing diagonal constraints, have a decidable emptiness problem.

References

[AAM06] Yasmina Abdeddaïm, Eugene Asarin, and Oded Maler. Scheduling with timed automata. *Theoretical Computer Science*, 354(2):272–300, 2006.

[ABG+08] S. Akshay, Benedikt Bollig, Paul Gastin, Madhavan Mukund, and K. Narayan Kumar. Distributed timed automata with independently evolving clocks. In *Proceedings of the 19th International Conference on Concurrency Theory (CONCUR 2008)*, volume 5201 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2008.

[ACM97] Eugene Asarin, Paul Caspi, and Oded Maler. A Kleene theorem for timed automata. In *Proceedings of the 12-th International Symposium in Logic in Computer System (LICS'97)*, pages 160–171. IEEE Computer Society Press, 1997.

[ACM02] Eugene Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *Journal of the Association for Computing Machinery*, 49:172–206, 2002.

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

[AM01] Yasmina Abdeddaïm and Oded Maler. Job-shop scheduling using timed automata. In *Proceedings of the 13-th International Conference on Computer-Aided Verification (CAV'01)*, volume 2102 of *Lecture Notes in Computer Science*, pages 478–492. Springer Verlag, 2001.

[AM02] Yasmina Abdeddaïm and Oded Maler. Preemptive job-shop scheduling using stopwatch automata. In *Proceedings of the 8-th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, volume 2280 of *Lecture Notes in Computer Science*, pages 113–126. Springer Verlag, 2002.

[BDFP04] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2-3):291–345, 2004.

[BH09] Béatrice Bérard and Serge Haddad. Interrupt timed automata. In *Proceedings of FoSSaCS'09*, volume 5504 of *Lecture Notes in Computer Science*, pages 197–211. Springer, 2009.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

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

[BP01] Patricia Bouyer and Antoine Petit. A kleene/büchi-like theorem for clock languages. *J. Autom. Lang. Comb.*, 7(2):167–186, 2001.

[BPT03] Patricia Bouyer, Antoine Petit, and Denis Thérien. An algebraic approach to data languages and timed languages. *Information and Computation*, 182(2):137–162, 2003.

[CGP99] Edmund Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. The MIT Press, 1999.

[Dim99] Catalin Dima. Kleene theorems for event-clock automata. In *Proceedings of FCT'99*, volume 1684 of *LNCS*, pages 215–225, 1999.

[Dim01] Catalin Dima. Real-time automata. *Journal of Automata, Languages and Combinatorics*, 6:3–23, 2001.

[Dim03] Catalin Dima. A nonarchimedian discretization for timed languages. In *Proceedings of the First International Workshop on Formal Modelling and Analysis of Timed Systems (FORMATS'03)*, volume 2791 of *Lecture Notes in Computer Science*, pages 161–181. Springer Verlag, 2003.

[Dim05] Catalin Dima. Timed shuffle expressions. In *Proceedings of the 16-th International Conference on Concurrency Theory (CONCUR'05)*, volume 3653 of *Lecture Notes in Computer Science*, pages 95–109. Springer Verlag, 2005.

[Dim09] Catalin Dima. Positive and negative results on the decidability of the model-checking problem for an epistemic extension of timed ctl. In *Proceedings of TIME'09*, pages 29–36. IEEE Computer Society, 2009.

[DL07] Catalin Dima and Ruggero Lanotte. Distributed time-asynchronous automata. In *Proceedings of the 4th International Colloquium on Theoretical Aspects of Computing (ICTAC 2007)*, volume 4711 of *Lecture Notes in Computer Science*, pages 185–200. Springer Verlag, 2007.

[DZ98] François Demichelis and Wieslaw Zielonka. Controlled timed automata. In *Proceedings of CONCUR'98*, volume 1466 of *Lecture Notes in Computer Science*, pages 455–469. Springer, 1998.

[FKPY07] Elena Fersman, Pavel Krcál, Paul Pettersson, and Wang Yi. Task automata: Schedulability, decidability and undecidability. *Inf. Comput.*, 205(8):1149–1172, 2007.

[FMPY06] Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Schedulability analysis of fixed-priority systems using timed automata. *Theor. Comput. Sci.*, 354(2):301–317, 2006.

[HKPV98] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata. *Journal of Computer Systems Science*, 57:94–124, 1998.

[HRS98] Thomas A. Henzinger, Jean-François Raskin, and Pierre-Yves Schobbens. The regular real-time languages. In *Proceedings of 25-th International Conference on Automata, Logics and Programming (ICALP'98)*, volume 1443 of *Lecture Notes in Computer Science*, pages 580–591. Springer Verlag, 1998.

[Kri99] Padmanabhan Krishnan. Distributed timed automata. *Electronic Notes in Theoretical Computer Science*, 28, 1999.

[OW03] Joël Ouaknine and James Worrell. Revisiting digitization, robustness, and decidability for timed automata. In *Proceedings of the 18-th International Symposium in Logic in Computer System (LICS'03)*, pages 198–207. IEEE Computer Society Press, 2003.

[PH98] Paritosh K. Pandya and Dang Van Hung. Duration calculus of weakly monotonic time. In *Proceedings of FTRTFT'98*, volume 1486 of *Lecture Notes in Computer Science*, pages 55–64. Springer, 1998.

[Wil94] Thomas Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In *Proceedings of FTRTFT'94*, volume 863 of *LNCS*, pages 694–715, 1994.

[WL05] Bozena Wozna and Alessio Lomuscio. A logic for knowledge, correctness, and real time. In *Proceedings of CLIMA V*, volume 3487 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2005.

[Yov98] Sergio Yovine. Model-checking timed automata. In *Lectures on Embedded Systems*, volume 1494 of *Lecture Notes in Computer Science*, pages 114–152. Springer Verlag, 1998.

[Zie87] Wieslaw Zielonka. Notes on finite asynchronous automata. *Informatique Théorique et Applications*, 21(2):99–135, 1987.