Timed shuffle expressions⊛

Cătălin Dima

Laboratoire d'Algorithmique, Complexité et Logique,
Université Paris XII – Val de Marne, 61 av. du Général de Gaulle, 94010 Créteil Cedex, France

Abstract. We show that stopwatch automata are equivalent to timed shuffle expressions, an extension of timed regular expressions with the shuffle operation. This implies that the emptiness problem for timed shuffle expressions is undecidable. The result holds for both timed state sequence semantics and timed event sequence semantics of automata and expressions.
Similarly to timed regular expressions, our timed shuffle expressions employ renaming. But we show that even when renaming is not used, shuffle regular expressions still have an undecidable emptiness problem. This solves in the negative a conjecture of Asarin on the possibility to use shuffle to define timed regular languages.
We also define a subclass of timed shuffle expressions which can be used to model preemptive scheduling problems. Expressions in this class are in the form $(E_1 \sqcup\!\sqcup \ldots \sqcup\!\sqcup E_n) \wedge E$, where E_i and E do not use shuffle. We show that emptiness checking within this class is undecidable too.

1 Introduction

Regular expressions are an important and convenient formalism for the specification of sets of discrete behaviors. Their connection to automata is one of the cornerstones of theoretical computer science, relating the class of behaviors recognizable by a finite-memory device to those that can be characterized as regular.

In the past decade several results have been lifted to the theory of timed systems over a continuous time domains. Several classes of regular expressions have been devised [4–7], but the connection between automata and regular expressions is less elegant than in classical automata theory. For example, the timed regular expressions of [4] need intersection and renaming in order to be equivalent to the timed automata of [1].

Eugene Asarin has recently asked [2] a series of questions whose answers would hopefully "substantially improve our understanding of the area" of timed systems. One of these questions is whether the shuffle of two timed regular languages is always regular. It was observed that, e.g. $5 \sqcup\!\sqcup 3 = 8$ and that $5a \sqcup\!\sqcup 3b$ is a timed regular language. A positive answer to this question might have helped the development of an alternative set of regular expressions for timed automata, with the hope that shuffle would eventually replace intersection and/or renaming.

In this paper we show that timed regular expressions (in the sense of [4] or [3]) extended with shuffle are equivalent to stopwatch automata [9], when they employ renaming. This result implies that they are strictly more powerful than timed regular

⊛ Partially supported by the PAI "Brancusi" no. 08797XL

expressions, at least because the emptiness problem for stopwatch automata is undecidable. We then show that even without renaming, timed regular expressions extended with shuffle have an undecidable emptiness problem. These results rely on the possiblity to encode non-difference constraints with a combination of shuffle and intersection. An example of such a combination gives a negative answer to the question of Asarin.

We also define a subclass of timed shuffle expressions which could be regarded as a "specification language" for preemptive scheduling problems. Our expressions, called *preemptive scheduling expressions*, are in the form $(E_1 \sqcup\!\sqcup \ldots \sqcup\!\sqcup E_n) \wedge E$, where E_i and E do not use shuffle. We show that emptiness checking within this class is undecidable too. This result complements results of [12] on the undecidability of emptiness checking for some class of timed automata augmented with preemptive jobs and different types of scheduling strategies. We note here that the expressive power of our preemptive scheduling expressions is not an issue in this paper, though it is an interesting question in itself.

Our proofs work for both state-based (i.e. signals or timed state sequences) and action-based (i.e. timed words) semantics. When presenting our undecidability result, we concentrate on state-based semantics, since preempting a job is somewhat equivalent to a "state change" in the system. But we also give an encoding of state-based expressions into action-based expressions, since the original conjecture of Asarin was stated on action-based semantics. However our undecidability results for action-based semantics rely on a "weakly monotonic semantics", that is, we allow two actions to occur at the same instant, but in a certain order. The restriction to strongly monotonic time is an open question.

The paper is divided as follows: in the second section we recall the timed event sequence (or *timed words*) framework and the timed state sequence (or *signals*) framework for the semantics of timed systems. Then, in the third section we recall the notion of stopwatch automata and the undecidability of their emptiness problem. The fourth section serves for the introduction of the timed shuffle expressions and for the proof of the Kleene theorem relating them to stopwatch automata. The fifth section presents the new undecidability results and the preemptive scheduling expressions, while the sixth section gives the translation of our results to action-based semantics. We end with a short section containing conclusions and further directions of study.

2 Timed languages

Timed event sequences and timed state sequences are the two alternative models for the behavior of timed systems. While timed event sequences put the accent on actions that a system is executing and on moments at which actions take place, timed state sequences put the accent on states in which the system is and on state durations.

A **signal** (or **timed state sequence**) is a finite sequence of pairs of symbols from Σ and nonnegative numbers. For example, the signal $(s, 1.2)\,(t, 1.3)\,(s, 0.1)$ denotes a behavior in which the *state* s holds for 1.2 time units, then is followed by the state t for another 1.3 time units, and ends with the state s again, which holds for 0.1 time units. For easier readability, we denote this signal as follows: $s^{1.3}t^{1.2}s^{0.1}$. The set of timed state sequences over Σ can be organized as a monoid, denoted $\mathsf{Sig}(\Sigma)$. Note that in this monoid, concatenation of identical signals amounts to the addition of their exponents, hence $s^{1.3}t^{0.3} \cdot t^{0.9}s^{0.1} = s^{1.3}t^{1.2}s^{0.1}$. And also $s^0 = \varepsilon$, the empty signal. The length $\ell(\omega)$ of a signal ω is the sum of all the numbers occurring in it, e.g. $\ell(s^{1.3}t^{1.2}s^{0.1}) = 1.2+1.3+0.1=2.6$. **Timed (signal) languages** are then sets of signals.

We will also work with **timed event sequences** (or **timed words**) which are finite sequences of nonnegative numbers and symbols from Σ. For example, the sequence $1.2\,a\,1.3\,b$ denotes a behavior in which an *action* a occurs 1.2 time units after the beginning of the observation, and after another 1.3 time units action b occurs. The set of timed words over Σ can be organized as a monoid w.r.t. concatenation; we denote this monoid as $\mathsf{TW}(\Sigma)$. Note that in this monoid, concatenation of two reals amounts to summation of the reals, hence, $a\,1.3 \cdot 1.7\,b = a(1.3+1.7)b = a\,3\,b$. The length $\ell(w)$ of a timed word w is the sum of all the reals in it, e.g. $\ell(1.2\,a\,1.3\,b) = 1.2+1.3 = 2.5$. **Timed event languages** are then sets of timed words.

Besides concatenation, we will be interested in the *shuffle* operation, which is the generalization of shuffle on Σ^*. Formally, given $w_1, w_2 \in \mathsf{Sig}(\Sigma)$

$$w_1 \sqcup\!\sqcup w_2 = \big\{u_1v_1 \dots u_nv_n \mid w_1 = u_1 \dots u_n, w_2 = v_1 \dots v_n\big\} \tag{1}$$

A shuffle operation with the same definition as above can be defined on timed words.

Shuffle can be extended as a set operation to languages: given $L_1, L_2 \subseteq \mathsf{Sig}(\Sigma)$ (or $L_1, L_2 \subseteq \mathsf{TW}(\Sigma)$) $L_1 \sqcup\!\sqcup L_2 = \bigcup\big\{w_1 \sqcup\!\sqcup w_2 \mid \sigma_1 \in L_1, \sigma_2 \in L_2\big\}$.

Another useful operation on timed languages is *renaming*: it simply replaces some symbols with some others, while keeping durations the same. The renaming of $a \in \Sigma$ with $b \in \Sigma$ is denoted $[a/b]$, For signals, renaming cannot delete symbols. An example of renaming on signals is $t^{2.5}u^{0.1} = [s/t]\big(s^{1.3}t^{1.2}u^{0.1}\big)$.

For timed words, we may also employ symbol deletion. The deletion of a symbol $a \in \Sigma$ is denoted $[a/\varepsilon]$. By abuse of notation (and of naming), we will call *renaming* also an operation in which some of the *action* symbols are deleted (but not time passage!). For example, $[a/c][b/\varepsilon]\big(1.3\,a\,1.2\,b\,0.1\,a\big) = 1.3\,c\,1.3\,c$.

3 Stopwatch automata

We recall here the definition of stopwatch automata [9], adapted such that they accept signals in our setting.

A **stopwatch automaton** [9] is a tuple $\mathcal{A} = (Q, \mathcal{X}, \Sigma, \eta, \lambda, \delta, Q_0, Q_f)$ where Q is a finite set of *states*, $\mathcal{X}$ is a finite set of *stopwatches*, Σ is a finite set of *state symbols*, $Q_0, Q_f \subseteq Q$ are sets of *initial*, resp. *final* states, $\lambda : Q \to \Sigma$ is the location labeling mapping, $\eta : Q \to \mathcal{P}(\mathcal{X})$ is a mapping assigning to each state the set of stopwatches that are active in that state. Finally, δ is a finite set of tuples (i.e. *transitions*), of the form (q, C, X, q'), where $q, q' \in Q$, $X \subseteq \mathcal{X}$, and C is a finite conjunction of *stopwatch constraints*. Stopwatch constraints that can be used in transitions are of the form $x \in I$, where $x \in \mathcal{X}$ and $I \subseteq [0, \infty[$ is an interval with integer (or infinite) bounds.

For each transition $(q, C, X, r) \in \delta$, the component C is called the *guard* of the transition, and X is called the *reset component* of the transition.

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

and

$$\theta = \big\{(q,v) \xrightarrow{\tau} (q,v') \mid v'_i = v_i + \tau,\ \forall i \in [n] \text{ with } x_i \in \eta(q), v'_i = v_i \text{ otherwise.}\big\}$$
$$\cup \big\{(q,v) \to (q',v') \mid \exists (q,C,X,q') \in \delta \text{ such that } v \models C \text{ and } \forall 1 \le i \le n, \quad (2)$$
$$\text{if } i \in X \text{ then } v'_i = 0 \text{ and if } i \notin X \text{ then } v'_i = v_i\big\}$$

In the line (2) of the definition of θ we say that the transition $q \xrightarrow{C,X} q' \in \delta$ *generates* the transition $(q,v) \to (q',v') \in \theta$.

Informally, the automaton can make time passage transitions (without changing location), in which all stopwatches *that are active in that location* advance by τ, and discrete transitions, in which location changes. The discrete transitions are enabled when the "current stopwatch valuation" v satisfies the guard C of a certain tuple $(q,C,X,q') \in \delta$, and when they are executed, the stopwatches in the reset component X are set to zero.

The **label** of a discrete transition $(q,v) \to (q',v')$ is ε, the empty sequence. The **label** of a time passage transition $(q,v) \xrightarrow{\tau} (q,v')$ is $\lambda(q)^\tau$.

A *run* in $\mathcal{T}(\mathcal{A})$ is a chain $\rho = \Big((q_{i-1},v_{i-1}) \xrightarrow{\xi_i} (q_i,v_i)\Big)_{1 \le i \le k}$ of transitions from θ, while a *run* in $\mathcal{A}$ is a chain of transitions $\rho' = \Big(q_{i-1} \xrightarrow{C_i,X-i} q_i\Big)_{1 \le i \le k'}$ in δ. The two runs ρ, ρ' are **associated** iff the i-th discrete transition in ρ is generated by the i-th transition of ρ'. An **accepting run** in $\mathcal{T}(\mathcal{A})$ is a run which starts in $\mathcal{Q}_0$, ends in $\mathcal{Q}_f$ and *does not end with a time passage transition*. An accepting run **accepts** a signal w iff w represents the formal concatenation of the labels of the transitions in the run.

The **language accepted by** $\mathcal{A}$ is then the set of signals which are accepted by some accepting run of $\mathcal{T}(\mathcal{A})$. The language accepted by $\mathcal{A}$ is denoted $L(\mathcal{A})$. Two timed automata are called **equivalent** iff they have the same language.

The Figure 1 gives an example of a stopwatch automaton. The active stopwatches in each location are identified as having derivative one; hence in location q_1 only stopwatch x is active. The language of this automaton is

$$\big\{s^{t_1}u^{t_2}\dots s^{t_{2k-1}}u^{t_{2k}}s^t \mid \sum t_{2i-1} \in [2,3], \sum t_{2i} \in [3,4[\,,\ t + \sum t_{2i-1} = 3\big\}$$

(recall that accepting runs cannot end with time passage transitions, hence the label ν of the final state may never occur in any accepted signal!)

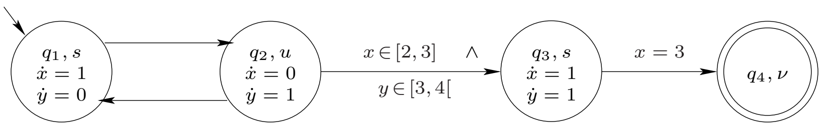

Fig. 1. A stopwatch automaton on signals.

A stopwatch automaton in which all stopwatches are active in each location is a *timed automaton*. In this case we will speak of *clocks* instead of stopwatches.

The *underlying untimed automaton* for a stopwatch automaton $\mathcal{A}$ is the automaton that keeps only the information regarding the transitions between states, and "forgets" all about stopwatch values and constraints. Hence, if $\mathcal{A} = (Q, \mathcal{X}, \Sigma, \eta, \lambda, \delta, Q_0, Q_f)$

then the underlying untimed automaton for $\mathcal{A}$ is $\hat{\mathcal{A}} = (Q, \emptyset, \Sigma, \eta, \lambda, \delta', Q_0, Q_f)$ where $(q, \mathsf{true}, \emptyset, r) \in \delta'$ if and only if $(q, C, X, r) \in \delta$ for some constraint C and subset of stopwatches $X \subseteq \mathcal{X}$.

Theorem 1 ([9]). *The emptiness problem for stopwatch automata is undecidable.*

The essential property that induces this undecidability is the fact that division by two, resp. doubling the value of a stopwatch can be simulated in stopwatch automata.

4 Timed shuffle expressions

In this section we introduce the class of timed shuffle expressions and prove their equivalence with stopwatch automata.

The syntax of **timed shuffle expressions** is the following:

$$E ::= s \mid E + E \mid E \cdot E \mid E \wedge E \mid E^* \mid \langle E \rangle_I \mid E \sqcup\!\sqcup E \mid [s/s']E$$

Here I is an interval and $s, s' \in \Sigma$. An expression not using shuffle is a **timed regular expression**.

The *semantics* of a timed shuffle expression (with renaming) is given by the following rules:

$$\begin{array}{rlrl}
\|s\| &= \{s^l \mid l \in \mathbb{R}_{\geq 0}\} & \|[s/u]E\| &= \big\{[s/u](w) \mid w \in \|E\|\big\} \\
\|E_1 + E_2\| &= \|E_1\| \cup \|E_2\| & \|E^*\| &= \|E\|^* \\
\|E_1 \wedge E_2\| &= \|E_1\| \cap \|E_2\| & \|\langle E \rangle_I\| &= \big\{w \in \|E\| \mid \ell(w) \in I\big\} \\
\|E_1 \cdot E_2\| &= \|E_1\| \cdot \|E_2\| & \|E_1 \sqcup\!\sqcup E_2\| &= \|E_1\| \sqcup\!\sqcup \|E_2\|
\end{array}$$

For example, the following expression is equivalent to the automaton in Figure 1:

$$[s_1/s]\Big(\Big(\big\langle \langle s_1 \rangle_{[2,3]} s \big\rangle_3 \sqcup\!\sqcup u\Big) \wedge \Big(\langle u \rangle_{[3,4[} s \sqcup\!\sqcup s_1\Big) \wedge (s_1 u)^* s\Big) \tag{3}$$

Theorem 2. *Timed shuffle expressions with renaming have the same expressive power as stopwatch automata.*

Proof. For the direct inclusion, the union, intersection, concatenation, star, time binding and renaming constructions from [4] can be easily extended to stopwatch automata. We will only give here the construction for the shuffle of two automata. The basic idea is to put together the two automata, making the control pass nondeterministically from one automaton to the other; moreover, to accept a timed word, the run must pass through an initial location and a final location in both automata.

So take two automata $\mathcal{A}_i = (Q_i, \mathcal{X}_i, \Sigma, \eta_i, \lambda_i, \delta_i, Q_0^i, Q_f^i)$ $(i = 1, 2)$. The automaton accepting $L(\mathcal{A}_1) \sqcup\!\sqcup L(\mathcal{A}_2)$ is then $\mathcal{A} = (Q, \mathcal{X}, \Sigma, \eta, \lambda, \delta, Q_0, Q_f)$ 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 $Q_f = Q_f^1 \times Q_f^2 \times \{1, 2\}$.
- $\lambda : Q \to \Sigma$ is defined by $\lambda(q_1, q_2, i) = \lambda_i(q_i)$, while $\eta : Q \to \mathcal{P}(\mathcal{X}_1 \cup \mathcal{X}_2)$ is defined by $\eta(q_1, q_2, i) = \eta_i(q_i)$.
- the transition relation is defined as follows:

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

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 the intersection of n one-stopwatch automata – similarly to the proof of the Kleene theorem for timed automata [4]. The second step shows how to associate a timed shuffle expression to an automaton with one stopwatch, by generalizing the construction of the expression (3) associated to the automaton in Figure 1.

The decomposition step requires a relabeling of the states of $\mathcal{A}$ such that two different states bear different labels. To do this, we simply replace Σ with Q as state labels, and put the identity function $id : Q \to Q$ as the state labeling function. Hence, if $\tilde{\mathcal{A}} = (Q, \mathcal{X}, Q, \eta, id, \delta, Q_0, Q_f)$ is the result of the above transformation, then $L(\mathcal{A}) = \lambda^{\#}(L(\tilde{\mathcal{A}})$ where $\lambda^{\#}$ is the renaming defined by $\lambda : Q \to \Sigma$.

Then we decompose $\tilde{\mathcal{A}}$ into n automata $\mathcal{A}_i = (Q, \{x_i\}, Q, \eta_i, id, \delta_i, Q_0, Q_f)$ having a single stopwatch; here $\eta_i(q) = \eta(q) \cap \{x_i\}$ and δ_i is a copy of δ in which the guard and the reset component of each tuple is a projection on $\{x_i\}$. Then

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

For the second step, suppose that $\mathcal{B}$ is an automaton with a single stopwatch in which all states have distinct state labels, $\mathcal{B} = (Q, \{x\}, Q, \eta, id, \delta, Q_0, Q_f)$. The following terminology will be used throughout the rest of this proof: we will speak of a state q as being *x-active* if $\eta(q) = x$; otherwise, we will say q is *x-inactive*. A run that passes through x-inactive states – excepting perhaps the starting and the ending state – is called a *x-inactive run*.

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

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

In this decomposition, $\mathcal{B}_1$ is a one-clock timed automaton, while $\mathcal{B}_2$ is an untimed automaton and $\hat{\mathcal{B}}$ is the underlying untimed automaton for $\mathcal{B}$. $\mathcal{B}_1$ 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 $\hat{\mathcal{B}}$ is to correctly connect the sequences of states in which x is active with those in which x is inactive. The ideas of this decomposition can be traced to the expression (3) associated to the automaton in Figure 1.

Consider then an x-inactive run $\rho = \big(q_{i-1} \xrightarrow{C_i,\emptyset} q_i\big)_{1 \le i \le k}$, in which x is never reset. Note that $C_i = (x \in I_i)$ for some nonnegative interval I_i. Throughout this run, the value of the stopwatch x remains unchanged, hence all the guards could be replaced with a single guard $x \in I_\rho := \bigcap_{1 \le i \le k} I_i$, which can be placed on any of the transitions. This implies that, in order to construct $\mathcal{B}_1$, we need to consider all sub-runs that contain only states inactive for x; then, for each such run, construct the intersection of the intervals on their guards.

Note further that we do not need to take into consideration runs with circuits: if ρ is a run from q to q' and $\overline{\rho}$ is another run that was constructed from ρ by adding a circuit in a state of ρ, then $I_\rho \supseteq I_{\overline{\rho}}$, hence a transition from q to q' labeled with $I_{\overline{\rho}}$ is superfluous. The consequence of this observation is that, given any two transitions $\tau, \tau' \in \delta$, the set of all non-superfluous intervals I_ρ associated to runs that start with τ and end in τ' can be computed by a Dijkstra-like algorithm. Let us denote then

$$\mathcal{I}_{\tau\tau'} = \big\{ I_\rho \mid \rho \text{ is an } x\text{-inactive run without circuits that starts with transition } \tau, \\ \text{ends with } \tau' \text{ and contains no transition resetting } x \,\big\}.$$

When $\tau = \tau' = q \xrightarrow{x\in I,\emptyset} q'$ we put $\mathcal{I}_{\tau\tau'} = \{I\}$.

The above remarks deal with x-inactive runs that do not pass through a transition resetting x. Taking into consideration resetting transitions can be done as follows: consider $\rho = \big(q_{i-1} \xrightarrow{C_i, X_i} q_i\big)_{1\leq i \leq k}$ an x-inactive run in which the first transition which resets x is $q_{j-1} \xrightarrow{C_j,\{x\}} q_j$. This run is unfeasible if there exists a guard $C_i : (x \in I_i)$ with $i > j$ for which $0 \notin I_i$. Hence, for all feasible runs ρ we can replace all C_i with $i>j$ with $x=0$ while all the guards C_i with $i \leq j$ could be replaced with $x \in \bigcap_{1\leq i\leq j} I_i$. Thence, given $q' \in Q$ and a transition $\tau \in \delta$, we do the following constructions:

1. Build $\mathsf{Reset}(q') = \big\{ \tau' \in \delta \mid \tau' = r \xrightarrow{x\in J,\{x\}} r' \text{ and } q' \text{ is reachable from } r'$ $\text{through an } x\text{-inactive run whose clock constraints all contain } 0.\big\}$

   This set is computable by backward reachability analysis on $\mathcal{B}$.
2. For each $\tau' = r \xrightarrow{x\in J,\{x\}} r' \in \mathsf{Reset}(q')$ and $\tau'' = r'' \xrightarrow{C,X} r \in \delta$, compute $\mathcal{I}_{\tau\tau''}$ as above.
3. Then compute $\mathcal{J}_{\tau\tau'} := \big\{ I \cap J \mid I \in \mathcal{I}_{\tau\tau''} \big\}$. where J is the interval labeling $\tau' = r \xrightarrow{x\in J,\{x\}} r'$.
4. If $\tau = \tau'$ then we put $\mathcal{J}_{\tau\tau'} = \big\{J\big\}$ (similarly to the fact that circuits can be ignored when computing $\mathcal{I}$).

The timed automaton $\mathcal{B}_1$ is then $\mathcal{B}_1 = \big(Q_1 \cup Q_f, \{x\}, Q, \eta_1, id, \delta_1, Q_0^1, Q_f\big)$ where

$$Q_1 = \big\{ q \in Q \mid q \text{ is } x\text{-active} \,\big\} \text{ and } \eta_1(q) = \{x\} \text{ for all } q \in Q_1 \cup Q_f$$
$$Q_0^1 = \big\{ q \in Q_1 \mid \exists \rho \; x\text{-inactive run starting in } q_0 \in Q_0 \text{ and ending in } q \text{ and} \\ \text{whose clock constraints all contain } 0 \big\}$$

$$\delta_1 = \big\{ q \xrightarrow{C,X} q' \in \delta \mid q, q' \in Q_1 \big\}$$
$$\cup \big\{ q \xrightarrow{x\in I,X} q' \mid q, q' \in Q_1, \exists \tau_1 = q \xrightarrow{x\in I_1,\emptyset} r_1 \in \delta, \tau_2 = r_2 \xrightarrow{x\in I_2,X} q' \in \delta \text{ s.t.} \\ \eta(r_1) = \eta(r_2) = \emptyset \text{ and } I \in \mathcal{I}_{\tau_1\tau_2} \big\} \quad (4)$$
$$\cup \big\{ q \xrightarrow{x\in I,\{x\}} q' \mid q, q' \in Q_1, \exists \tau_1 = q \xrightarrow{x\in I_1,X_1} r_1 \in \delta, \tau_2 = r_2 \xrightarrow{C,z,\{x\}} r_3, \\ \tau_2 \in \mathsf{Reset}(q') \text{ s.t. } \eta(r_1) = \eta(r_2) = \emptyset, \tau_2 \in \mathsf{Reset}(q') \text{ and } I \in \mathcal{J}_{\tau_1\tau_2} \big\} \quad (5)$$
$$\cup \big\{ q \xrightarrow{x\in I,\emptyset} q_f \mid q_f \in Q_f, \exists \tau_1 = q \xrightarrow{x\in I,X_1} q_1, \tau_2 = q_2 \xrightarrow{x\in J,X_2} q_3 \in \delta \text{ and} \\ \text{either } I \in \mathcal{I}_{\tau_1\tau_2} \text{ and } q_3 = q_f \text{ or } I \in \mathcal{J}_{\tau_1\tau_2} \text{ and } \tau_2 \in \mathsf{Reset}(q_f) \big\} \quad (6)$$

Note that the components 4 and 5, are used for assembling pieces of x-active parts in an accepting run in $\mathcal{B}$.

The timed automaton $\mathcal{B}_2$ is then $\mathcal{B}_2 = \big(Q_2, \emptyset, Q_2, \eta_2, id, \delta_2, Q_0^2, Q_f^2\big)$ where

$$Q_2 = \big\{q \in Q \mid \eta(q) = \emptyset\big\} \text{ and } \eta_2(q) = \emptyset \text{ for all } q \in Q_2$$
$$Q_0^2 = \big\{q \in Q_2 \mid q \in Q_0 \text{ or there exists an } x\text{-active run } \rho \text{ starting in } Q_0 \text{ and ending in } q\big\}$$
$$Q_f^2 = \big\{q \in Q_2 \mid q \in Q_f \text{ or there exists an } x\text{-active run } \rho \text{ starting in } q \text{ and ending in } Q_f\big\}$$
$$\delta_2 = \big\{q \xrightarrow{\mathsf{true},\emptyset} q' \in \delta \mid q, q' \in Q_2\big\} \cup$$
$$\big\{q \xrightarrow{\mathsf{true},\emptyset} q' \mid \text{ there exists an } x\text{-active run } \rho \text{ starting in } q \text{ and ending in } q' \big\}$$

Hence, $\mathcal{B}_2$ keeps track of the x-inactive parts of each accepting run of $\mathcal{B}$. To correctly combine runs in $\mathcal{B}_1$ with runs in $\mathcal{B}_2$ we further intersect their shuffle with $L(\hat{\mathcal{B}})$, the language of the underlying untimed automaton for $\hat{\mathcal{B}}$. This final intersection forbids incorrect "plugging" of parts of x-active runs, as provided by $\mathcal{B}_1$, with parts of x-inactive runs, as found in $\mathcal{B}_2$.

The Kleene theorem for timed automata [3] assures then the existence of timed regular expressions equivalent to each of the three automata ($\mathcal{B}_1$. $\mathcal{B}_2$ and $\hat{\mathcal{B}}$), fact which ends our proof. □

Note that the equivalence in Theorem 2 is effective, i.e. all the constructions in the proof are algorithmic. This ensures then the following

Corollary 1. *The emptiness problem for timed shuffle expressions with renaming is undecidable.*

5 Timed shuffle expressions without renaming are undecidable

The technique used in the papers [3, 10] can be adapted to show that intersection and renaming are necessary for the Theorem 2 to hold. However we will be interested here in a different problem: can we diminish the expressive power of the timed shuffle expressions (without giving up the shuffle!) such that they be comparable to timed automata? And the first natural question is to compare timed shuffle expressions without renaming and timed regular expressions.

In this section we will show that the emptiness problem remains undecidable even for s timed shuffle expressions without renaming. Recall that the essential property that gives the undecidability is the fact that we may "double" the value of a stopwatch. We will adapt this property to timed shuffle expressions without renaming.

Remark 1. We start with an example showing that timed shuffle expressions may induce more general linear constraints than those induced by timed regular expressions. Namely, for any timed word in the semantics of the following expression:

$$E_{1/2} = \big\langle \mathbf{ps} \big\rangle_1 \mathbf{uvz} \wedge \big(\mathbf{p}\big\langle \mathbf{suz} \big\rangle_1 \sqcup\!\sqcup \mathbf{v}\big) \wedge \mathbf{ps}\big\langle \mathbf{uv} \big\rangle_1 \mathbf{z} \wedge \mathbf{psu}\big\langle \mathbf{vz} \big\rangle_1$$

the duration of the state symbol $\mathbf{p}$ represents twice the duration of the state symbol $\mathbf{u}$:

$$\|E_{1/2}\| = \big\{\mathbf{p}^{t_1}\mathbf{s}^{t_2}\mathbf{u}^{t_3}\mathbf{v}^{t_4}\mathbf{z}^{t_5} \mid t_1 = 2t_3, t_1 + t_2 = 1 = t_3 + t_4, t_3 = t_5\big\} \quad (7)$$

Note that t_2 is used as a reference for asserting that t_1 and $t_3 + t_5$ must be equal, and similarly for t_4, which asserts that t_3 and t_5 are equal. On the other hand, t_5 is used as a "divisor".

The main result of this section is the following:

Theorem 3. *The emptiness problem for timed shuffle expressions without renaming is undecidable.*

Proof. The proof goes by reduction of the problem of the existence of a terminating computation of a Minsky machine [11]. We will show that, for each 2-counter (Minsky) machine M, the computation of M can be encoded into a timed shuffle expression E. To this end, we encode the value of each counter of the machine M in some configuration as the duration of a certain state symbol. The value of the counter x, denoted x too, will be encoded by the signal $\mathbf{s}^{2^{-x}} = \mathbf{s}^{\frac{1}{2^x}}$ and similarly, y is encoded by $\mathbf{u}^{2^{-y}}$. Then, decrementing the counter x amounts to "doubling the length" of the state symbol $\mathbf{s}$ whereas incrementing x amounts to "dividing the length" of $\mathbf{s}$ by two, along the idea presented in the remark 1.

The encoding of a configuration (q, x, y) will consist of the set of timed state sequences of the type $q^{l_0} s_1^{l_1} \mathbf{s}^{2^{-x}} u_1^{l_2} \mathbf{u}^{2^{-y}}$ where $s_1, u_1, \mathbf{s}, \mathbf{u}$ are distinct symbols ($i \in [1 \ldots 3]$) and l_0, l_1, l_2 are some nonnegative numbers.

So suppose the 2-counter machine is $M = (Q, \theta, q_0, Q_f)$ where

$$\theta \subseteq Q \times \left\{x^{++}, y^{++}, x^{--}, y^{--}, x = 0?, y = 0?\right\} \times Q$$

Remind that we may consider M being *deterministic*, in a sense that implies that between two states q, r, at most one transition could occur in δ.

We will first associate to each transition $\tau = (q, op, r) \in \delta$ four timed regular expressions, $E^x_{(q,op,r)}$, $E^y_{(q,op,r)}$, $E^0_{(q,op,r)}$ and $E^\wedge_{(q,op,r)}$, with the aim to encode in the following expression:
$E_{(q,op,r)} := (E^x_{(q,op,r)} \sqcup\!\sqcup E^y_{(q,op,r)} \sqcup\!\sqcup E^0_{(q,op,r)}) \wedge E^\wedge_{(q,op,r)}$ the sequentialization of an encoding of a configuration (q, x, y) before taking τ with an encoding of the resulting configuration (q, x', y'), the result of τ. The desired expressions, for $op = x^{++}$, are:

$$E^x_{(q,x^{++},r)} = \langle q\rangle_1 s_1 \big\langle \mathbf{s}\langle r\rangle_1 s_1\big\rangle_3 \mathbf{s} s_3 \Big(\big\langle \mathbf{s} s_3\big\rangle_2 \langle r\rangle_1 \mathbf{s} \wedge \mathbf{s}\big\langle s_3 r \mathbf{s}\big\rangle_3\Big)$$
$$E^y_{(q,x^{++},r)} = \langle q\rangle_1 u_1 \big\langle \mathbf{u}\langle r\rangle_1 u_1\big\rangle_3 \big\langle \mathbf{u}\langle r\rangle_1 u_1\big\rangle_3 \mathbf{u} \wedge q u_1 \mathbf{u} r \big\langle u_1 \mathbf{u}\big\rangle_2 r \big\langle u_1 \mathbf{u}\big\rangle_2$$
$$E^0_{(q,x^{++},r)} = \langle q\rangle_1 s_2 s_3 u_2 u_3 \langle r\rangle_1 s_2 u_2 u_3 \langle r\rangle_1 s_1$$
$$E^\wedge_{(q,x^{++},r)} = \langle q\rangle_3 s_1 \mathbf{s} s_2 s_3 u_1 \mathbf{u} u_2 u_3 \langle r\rangle_3 \Big(\big\langle s_1 \mathbf{s} s_2\big\rangle_2 s_3 \wedge s_1 \mathbf{s}\big\langle s_2 s_3\big\rangle_2\Big) u_1 \mathbf{u} u_2 u_3 \langle r\rangle_3 s_1 \mathbf{s} u_1 \mathbf{u}$$

Note that if $q^3 s_1^{l_1} \mathbf{s}^l s_2^{l_2} s_3^{l_3} u_1^{m_1} \mathbf{u}^m u_2^{m_2} u_3^{m_3} r^3 s_1^{l_4} \mathbf{s}^{l'} s_2^{l_5} s_3^{l_6} u_1^{m_4} \mathbf{u}^{m'} u_2^{m_5} u_3^{m_6} r^3 s_1^{l_7} \mathbf{s}^{l''} u_1^{m_7} \mathbf{u}^{m''}$ belongs to $\|E_{(q,x^{++},r)}\|$, then

- $l + l_4 = l_4 + l' + l_5 = 2$ (by $E^x_{(q,x^{++},r)}$ and $E^\wedge_{(q,x^{++},r)}$).
- $l' + l_6 = l_5 + l_6 = l_6 + l'' = 2$ (again by $E^x_{(q,x^{++},r)}$ and $E^\wedge_{(q,x^{++},r)}$), hence $l = 2l' = 2l''$.
- $m + m_4 = m_4 + m' = m' + m_7 = m_7 + m'' = 2$ (by $E^y_{(q,x^{++},r)}$), hence $m = m' = m''$.

Hence if $q^3 s_1^{l_1}\mathbf{s}^l u_1^{m_1}\mathbf{u}^m$ encodes some configuration (q,x,y) then $q^3 s_1^{l_7}\mathbf{s}^{l''} u_1^{m_7}\mathbf{u}^{m''}$ encodes $(r,x+1,y)$, which is the result of the transition (q,op,r).

The subexpressions for $E_{(q,x--,r)}$ are:

$$E^x_{(q,x--,r)} = \langle q\rangle_1 \big\langle s_1 s_2\big\rangle_2 \langle r\rangle_1 \Big(s_1 \big\langle \mathbf{s} r s_1\big\rangle_3 \mathbf{s}\Big)$$
$$E^y_{(q,x--,r)} = \langle q\rangle_1 u_1 \big\langle \mathbf{u}\langle r\rangle_1 u_1\big\rangle_3 \big\langle \mathbf{u}\langle r\rangle_1 u_1\big\rangle_3 \mathbf{u} \wedge q u_1 \mathbf{u} r \big\langle u_1\mathbf{u}\big\rangle_2 \langle r\rangle_1 \big\langle u_1 \mathbf{u}\big\rangle_2 r \big\langle u_1\mathbf{u}\big\rangle_2$$
$$E^0_{(q,x--,r)} = \langle q\rangle_1 s_2 s_3 \mathbf{u} u_3 \langle r\rangle_1 s_2 s_3 u_2 u_3 \langle r\rangle_1$$
$$E^\wedge_{(q,x--,r)} = \langle q\rangle_3 \Big(\big\langle s_1 \mathbf{s}\big\rangle_2 s_2 \big\langle s_3 u_1 \mathbf{u} u_2 u_3\big\rangle_4 \langle r\rangle_3 \big\langle s_1\mathbf{s}\big\rangle_2 \wedge s_1 \big\langle \mathbf{s} s_2 s_3\big\rangle_2 \big\langle u_1 \mathbf{u} u_2 u_3 r s_1\big\rangle_7 \mathbf{s}\Big)$$
$$\cdot\, s_2 s_3 u_1 \mathbf{u} u_2 u_3 \langle r\rangle_3 s_1 \mathbf{s} u_1 \mathbf{u}$$

Note that if $q^3 s_1^{l_1}\mathbf{s}^l s_2^{l_2} s_3^{l_3} u_1^{m_1}\mathbf{u}^m u_2^{m_2} u_3^{m_3} r^3 s_1^{l_4}\mathbf{s}^{l'} s_2^{l_5} s_3^{l_6} u_1^{m_4}\mathbf{u}^{m'} u_2^{m_5} u_3^{m_6} r^3 s_1^{l_7}\mathbf{s}^{l''} u_1^{m_7}\mathbf{u}^{m''}$ belongs to $\|E_{(q,x++,r)}\|$ then

- $l_1 + l = l_1 + l_2 = 2$ (by $E^x_{(q,x--,r)}$ and $E^\wedge_{(q,x--,r)}$), hence $l = l_2$.
- $m + m_4 = m_4 + m' = m' + m_7 = m_7 + m'' = 2$ (as implied by $E^y_{(q,x--,r)}$), hence $m = m' = m''$.
- $l + l_2 + l_3 = l_4 + l' = l' + l_7 = l_7 + l'' = 2$ and $l_3 + m_1 + m + m_2 + m_3 = m_1 + m + m_2 + m_3 + l_4 = 4$ (as implied by $E^x_{(q,x--,r)}$ and $E^\wedge_{(q,x--,r)}$), hence $l + l_2 = 2l = l' = l''$.

Finally, the subexpressions for $E_{(q,x=0?,r)}$ are the following:

$$E^x_{(q,x=0?,r)} = \langle q\rangle_1 s_1 \langle \mathbf{s}\rangle_1 \langle r\rangle_1 s_1 \langle \mathbf{s}\rangle_1 \langle r\rangle_1 s_1 \langle \mathbf{s}\rangle_1$$
$$E^y_{(q,x=0?,r)} = \langle q\rangle_1 u_1 \big\langle \mathbf{u}\langle r\rangle_1 u_1\big\rangle_3 \big\langle \mathbf{u}\langle r\rangle_1 u_1\big\rangle_3 \mathbf{u} \wedge q u_1 \mathbf{u} r \big\langle u_1 \mathbf{u}\big\rangle_2 \langle r\rangle_1 \big\langle u_1\mathbf{u}\big\rangle_2$$
$$E^0_{(q,x=0?,r)} = \langle q\rangle_1 s_2 s_3 u_2 u_3 \langle r\rangle_1 s_2 s_3 u_2 u_3 \langle r\rangle_1$$
$$E^\wedge_{(q,x=0?,r)} = \langle q\rangle_3 s_1 \mathbf{s} s_2 s_3 u_1 \mathbf{u} u_2 u_3 \langle r\rangle_3 \Big(\big\langle s_1 \mathbf{s} s_2\big\rangle_2 s_3 \wedge s_1 \mathbf{s} \big\langle s_2 s_3\big\rangle_2 \Big) u_1 \mathbf{u} u_2 u_3 \langle r\rangle_3 s_1 \mathbf{s} u_1 \mathbf{u}$$

Along the same lines, we want to construct an expression $E = \big(E_x \sqcup E_y \sqcup E_0\big) \wedge E_\wedge$, such that E_x gives the (encoding of the) projection of the run of M onto the counter x, E_y does the same for y, E_0 inserts some "locally unused" symbols, while $E_\wedge$ ensures that E_x and E_y apply synchronously the same transition of M.

Let us denote

$$E^x_\delta = \sum\nolimits_{(q,op,r)\in\delta} E^x_{(q,op,r)}, \qquad E^y_\delta = \sum\nolimits_{(q,op,r)\in\delta} E^y_{(q,op,r)},$$
$$E^0_\delta = \sum\nolimits_{(q,op,r)\in\delta} E^0_{(q,op,r)}, \qquad E^\wedge_\delta = \sum\nolimits_{(q,op,r)\in\delta} E^\wedge_{(q,op,r)}$$

An important remark to make here is that for all $\sigma \in \Big\|\big(E^x_\delta \sqcup E^y_\delta \sqcup E^0_\delta\big) \wedge E^\wedge_\delta\Big\|$ we have in fact that $\sigma \in \Big\|\big(E^x_{(q,op,r)} \sqcup E^x_{(q,op,r)} \sqcup E^x_{(q,op,r)}\big) \wedge E^x_{(q,op,r)}\Big\|$ for *the same* $(q,op,r)\in\delta$. (Remind that the Minsky machine was deterministic.)

To see this, note that $E^\wedge_\delta$ requires that σ contain only two state symbols $q, r \in Q$, both of which last for 3 time units. But then, if E^x_δ, E^y_δ and E^0_δ choose different transitions, i.e., if we pick $\sigma_1 \in E^x_{(q_1,op_1,r_1)}$, $\sigma_2 \in E^y_{(q_2,op_2,r_2)}$, $\sigma_3 \in E^0_{(q_3,op_3,r_3)}$, where at least two of the three transitions are distinct, then $\sigma_1 \amalg \sigma_2 \amalg \sigma_3$ contains three distinct state symbols of non-zero length from the set $\{(q_1)^1, (q_2)^1, (q_3)^1, (r_1)^1, (r_2)^1, (r_3)^1\}$. Hence, $\sigma_1 \amalg \sigma_2 \amalg \sigma_3 \notin \|E^\wedge_\delta\|$.

Thence, the expression $\Big(\big(E^x_\delta\big)^* \amalg \big(E^y_\delta\big)^* \amalg \big(E^0_\delta\big)^*\Big) \wedge \big(E^\wedge_\delta\big)^*$ encodes sequences of arbitrary transitions in M, but imposes no connection between two consecutive transitions either by state symbols or by counter values. We will then have to add more constraints to the subexpressions E^x_δ and E^y_δ, in order to ensure proper "propagation" of states and counter values.

For each $q \in Q$, consider the following expressions:

$$E^x_q = \langle q\rangle_1(\varepsilon + s_1)\Big(\big\langle \mathbf{s}\langle q\rangle_1 s_1\big\rangle_3(\mathbf{s} + s_2) \wedge \mathbf{s}q\big\langle s_1(\mathbf{s} + s_2)\big\rangle_2\Big)$$
$$E^x_q = \langle q\rangle_1(\varepsilon + u_1)\Big(\big\langle \mathbf{u}\langle q\rangle_1 u_1\big\rangle_3(\mathbf{u} + u_2) \wedge \mathbf{u}q\big\langle u_1(\mathbf{u} + u_2)\big\rangle_2\Big)$$

These expressions will be essential in asserting correct connection of intermediary states and counter values. Note that, for example, in each $\sigma = q^1\mathbf{s}^m q^1 s_1^{l_2}\mathbf{s}^{m'} \in \|E^x_q\|$, we have that $m = m'$, and similarly with $\sigma' = q^1 s_1^{l_1}\mathbf{s}^m q^1 s_1^{l_2} s_2^{m'} \in \|E^x_q\|$, and all the other types of signals in $\|E^x_q\|$.

Let us denote further

$$E_x = \big(E^x_\delta\big)^* \wedge q_0 s_1 \mathbf{s}\Big(\sum\nolimits_{q\in Q} q s_1 \mathbf{s}(s_3 + \varepsilon)E^x_q\Big)^* \cdot \sum\nolimits_{q\in Q_f} q s_1 \mathbf{s}(\varepsilon + s_3)q(\varepsilon + s_1)\mathbf{s}$$
$$E_y = \big(E^y_\delta\big)^* \wedge q_0 u_1 \mathbf{u}\Big(\sum\nolimits_{q\in Q} q u_1 \mathbf{u}(s_3 + \varepsilon)E^y_q\Big)^* \cdot \sum\nolimits_{q\in Q_f} q u_1 \mathbf{u}(\varepsilon + u_3)q(\varepsilon + u_1)\mathbf{u}$$
$$E_0 = \big(E^0_\delta\big)^*, \qquad E_\wedge = \big(E^\wedge_\delta\big)^*$$

Observe now that, in the expression E_x, we ensure correct connection of the intermediary states of M and of the counter values for x. This remark follows by a case study for each of the possibilities of having two successive transitions. For example, if $\sigma_1\sigma_2, \dots, \sigma_k \in E_x$ with $\sigma_i = q_i^{l_1^i} s_1^{l_2^i}\mathbf{s}^{m_i} r_i^{l_3^i} s_1^{l_4^i}\mathbf{s}^{m'_i} s_3^{l_5^i} r_i^{l_6^i} s_1^{l_7^i}\mathbf{s}^{m''_i} \in \|E^x_\delta\|$ for $1 \le i \le k$, then we must have that $r_{i-1} = q_i$ and $m'_{i-1} = m''_{i-1} = m_i$, that is, two consecutve signals represent part of the encoding of two consecutive transitions that share the intermediary state and the counter value for x.

Another example is when some σ_i is of the form $q_i^{l_1^i} s_1^{l_2^i} s_2^{m_i} r_i^{l_3^i} s_1^{l_4^i}\mathbf{s}^{m'_i} r_i^{l_6^i} s_1^{l_7^i}\mathbf{s}^{m''_i} \in \|E^x_\delta\|$. This situation occurs when $\sigma_i \in \big\|E^x_{(q_{i-1},x^{--},q_i)}\big\|$. We have again $r_{i-1} = q_i$ and $m'_{i-1} = m_i$, but this time it is the length of s_2 in σ_i that copies the duration of the last $\mathbf{s}$ in σ_{i-1}. But this is ok, if we recall from the definition of $E_{(q,x^{--},r)}$ that the length of the first s_2 and the length of the first $\mathbf{s}$ in σ_i are equal. Hence, again two consecutive signals represent part of the encoding of two consecutive transitions that share the intermediary state and the counter value for x. All the other cases lead to similar conclusions about the correct connection of intermediary states and counter values.

The expression that encodes the (unique!) run of M is the following:

$$E_\rho = \big(E_x \amalg E_y \amalg E_0\big) \wedge E_\wedge \tag{8}$$

Then $\|E_\rho\| = \emptyset$ iff M does not have a finite run. $\square$

The particularity of the proof of Theorem 3 suggests us that shuffle expressions without renaming are not the only interesting strict subclass of expressions that have an undecidable emptiness problem.

Preemptive scheduling expressions are timed shuffle expressions of the form

$$E \wedge (E_1 \sqcup\!\sqcup \ldots \sqcup\!\sqcup E_k)$$

where $E, E_1, \ldots, E_k$ are *timed regular expressions* – i.e. do not contain shuffle (but may contain renaming). Their name comes from the fact that $E_1, \ldots, E_k$ may be considered as expressions defining the behavior of some (preemptive) jobs, encapsulating duration constraints within each job, whereas E can be regarded as an expression embodying overall timing constraints for each job and the scheduling policy.

As an example, consider the following expressions:

$$E_1 = \big(s_1 \langle s_2 s_3 \rangle_2\big)^*, \quad E_2 = \big(u_1 \langle u_2 u_3 \rangle_2\big)^*$$
$$E = \big(\langle s_1 s_2 \rangle_2 (s_3 + u_1 + u_2 + u_3)^*\big)^* \wedge \big(\langle u_1 (s_1 + s_2 + s_3)^* u_2 \rangle_2 (s_1 + s_2 + s_3 + u_3)^*\big)^*$$

The expression $E \wedge (E_1 \sqcup\!\sqcup E_2)$ can be regarded as a specification of (the solution of) a two-job scheduling problem, in which

1. In the job E_1, the duration of s_2 plus s_3 equals 2 time units;
2. In the job E_2, the duration of u_2 plus u_3 equals 2 time units;
3. The part $s_1 s_2$ of E_1 cannot be preempted, hence has higher priority over E_2 and must be executed within 2 time units of the starting of signal state s_1.
4. The part $u_1 u_2$ of E_2 must be executed within 2 time units of the starting of signal state u_1, regardless of the interruptions.

The proof of Theorem 3 implies also the following:

Theorem 4. *Preemptive shuffle expressions have an undecidable emptiness problem.*

As we may see from the proof of Theorem 3, expressions with three "components" – that is, expressions of the form $E \wedge (E_1 \sqcup\!\sqcup E_2 \sqcup\!\sqcup E_3)$ – already have an undecidable emptiness problem. It is an open question whether two components would suffice.

6 Action-based semantics for automata and expressions

Throughout this section we show that all the results in this paper also apply to automata and expressions with timed word semantics.

Let us review in more detail the definition of stopwatch automata with action semantics: a **stopwatch automaton** is a tuple $\mathcal{A} = (Q, \mathcal{X}, \Sigma, \eta, \delta, Q_0, Q_f)$ where Q, Σ, $\mathcal{X}$, η, Q_0 and Q_f have the same meaning as in Section 3, while δ is a finite set of tuples (i.e. *transitions*), of the form (q, C, a, X, q'), where $q, q' \in Q$, $X \subseteq \mathcal{X}$, $a \in \Sigma$ and C is a finite conjunction of stopwatch constraints.

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

$$\theta = \big\{(q, v) \xrightarrow{\tau} (q, v') \mid v'_i = v_i + \tau,\ \forall i \in [n] \text{ with } x_i \in \eta(q), v'_i = v_i \text{ otherwise.}\big\}$$
$$\cup \big\{(q, v) \xrightarrow{a} (q', v') \mid \exists (q, C, a, X, q') \in \delta \text{ such that } v \models C \text{ and for all } i \in [n],$$
$$\text{if } i \in X \text{ then } v'_i = 0 \text{ and if } i \notin X \text{ then } v'_i = v_i\big\}$$

The **label** of a discrete transition $(q, v) \xrightarrow{a} (q', v')$ is the symbol a that lies on the arrow. The **label** of a time passage transition $(q, v) \xrightarrow{\tau} (q, v')$ is τ.

A *run* in $\mathcal{T}(\mathcal{A})$ is a chain $\rho = \Big((q_{i-1}, v_{i-1}) \xrightarrow{\xi_i} (q_i, v_i)\Big)_{1\leq i\leq k}$ of transitions from θ, while 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 two runs ρ, ρ' are **associated** iff the i-th discrete transition in ρ is generated by the i-th transition of ρ'. An **accepting run** in $\mathcal{T}(\mathcal{A})$ is a run which starts in $\mathcal{Q}_0$, ends in $\mathcal{Q}_f$ and *does not end with a time passage transition*. An accepting run **accepts** a timed word w iff w represents the formal concatenation of the labels of the transitions in the run.

The Figure 2 gives an example of a stopwatch automaton whose language is

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

Recall that the last transition in an accepting run cannot be a time passage transition and hence the automaton cannot spend any time in the last state q_4 when accepting a timed word.

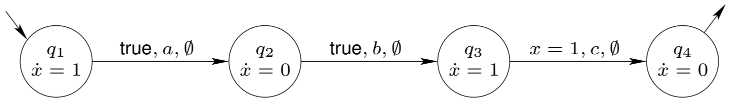

Fig. 2. A stopwatch automaton with action semantics.

The syntax of **timed action 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 I is an interval, $a \in \Sigma$ and $b \in \Sigma \cup \{\varepsilon\}$.

Their semantics follows the same compositional rules as the semantics of timed shuffle expressions over signals, with the only difference on the rules on atoms, which, for timed action shuffle expressions are the following:

$$\|a\| = \big\{a\big\}, \quad \|\underline{\mathbf{t}}\| = \big\{t \mid t \in \mathbb{R}_{\geq 0}\big\}$$

For example, the expression $E_1 = [z_1/\varepsilon][z_2/\varepsilon]\big((\langle z_1\underline{\mathbf{t}}az_1\underline{\mathbf{t}}c\rangle_1 \sqcup\!\sqcup z_2\underline{\mathbf{t}}b) \wedge z_1\underline{\mathbf{t}}az_2\underline{\mathbf{t}}bz_1\underline{\mathbf{t}}c\big)$ represents the language of the automaton $\mathcal{A}$ from Figure 2. Note the need to "duplicate" each transition label: if we do not make use of 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. That is, the following shuffle expression is not equivalent to the automaton in Figure 2:

$$E_2 = \big(\langle\underline{\mathbf{t}}a\underline{\mathbf{t}}c\rangle_1 \sqcup\!\sqcup \underline{\mathbf{t}}b\big) \wedge \underline{\mathbf{t}}a\underline{\mathbf{t}}b\underline{\mathbf{t}}c$$

(E_2 was obtained from E_1 by removing all occurrences of z_1 and z_2.) This expression is equivalent to the timed expression $E_3 = \langle\underline{\mathbf{t}}a\underline{\mathbf{t}}b\underline{\mathbf{t}}c\rangle_{]1,\infty[}$, which is obviously not what is needed. The problem lies in the fact that the duration before b in the subexpression $\underline{\mathbf{t}}b$ must not "mix" with the other durations. The use of the additional symbols z_1 and z_2 is essential in forbidding this mixing.

In order to compare the expressive power of state based and action based semantics, note that the "division" expression from Identity 7 can be converted to action-based semantics:

$$E^a_{1/2} = \langle a\underline{\mathbf{t}}\hat{a}\, b\underline{\mathbf{t}}\hat{b}\rangle_1 c\underline{\mathbf{t}}\hat{c}\, d\underline{\mathbf{t}}\hat{d}\, e\underline{\mathbf{t}}\hat{e} \wedge \big(a\underline{\mathbf{t}}\hat{a}\langle b\underline{\mathbf{t}}\hat{b}\, c\underline{\mathbf{t}}\hat{c}\, e\underline{\mathbf{t}}\hat{e}\rangle_1 \sqcup\!\sqcup\, d\underline{\mathbf{t}}\hat{d}\big) \wedge$$
$$a\underline{\mathbf{t}}\hat{a}\, b\underline{\mathbf{t}}\hat{b}\langle c\underline{\mathbf{t}}\hat{c}\, d\underline{\mathbf{t}}\hat{d}\rangle_1 e\underline{\mathbf{t}}\hat{e} \wedge a\underline{\mathbf{t}}\hat{a}\, b\underline{\mathbf{t}}\hat{b}\, c\underline{\mathbf{t}}\hat{c}\langle d\underline{\mathbf{t}}\hat{d}\, e\underline{\mathbf{t}}\hat{e}\rangle_1 \quad (9)$$

Here, the duration of the time interval between a and $\hat{a}$ represents twice the duration of the interval between c and $\hat{c}$:

$$\|E^a_{1/2}\| = \big\{at_1\hat{a}\, bt_2\hat{b}\, ct_3\hat{c}\, dt_4\hat{d}\, et_5\hat{e} \mid t_1 = 2t_3, t_1 + t_2 = 1 = t_3 + t_4, t_3 = t_5\big\}$$

Given a set of symbols Σ, consider a copy $\hat{\Sigma}$ of it, and denote $\hat{s}$ the copy of the symbol $s \in \Sigma$. Consider then the following morphism $\varphi : \mathsf{Sig}(\Sigma) \to \mathsf{TW}(\Sigma)$, defined by

$$\varphi(s^\alpha) = st^\alpha\hat{s}, \quad \varphi(w_1w_2) = \varphi(w_1)\varphi(w_2) \text{ for all } w_1, w_2 \in \mathsf{Sig}(\Sigma)$$

Then, given any stopwatch automaton $\mathcal{A}$ with signal semantics, we may associate to it a stopwatch automaton $\mathcal{B}$ with action semantics such that $L(\mathcal{B}) = \varphi\big(L(\mathcal{A})\big)$. A similar result can be provided for expressions E with signals semantics, by constructing an expression with action semantics E' such that $\|E'\| = \varphi\big(\|E\|\big)$.

We give here the construction for shuffle expressions, since the respective construction for automata can be derived by the Kleene theorem. This construction is straightforward: given a timed shuffle expression over signals E, we replace each atom $a \in \Sigma$ with $a\underline{\mathbf{t}}\hat{a}$. We denote the resulting expression $\varphi(E)$.

Remark 2. Note that $\big\|\varphi(E)\big\| \neq \emptyset$ iff $\big\|E\big\| \neq \emptyset$ and that, if we apply this construction to an expression without renaming or to a preemptive scheduling expression, the resulting expression remains in the same class.

Also note that $\varphi\Big([\mathbf{p}/a][\mathbf{s}/b][\mathbf{u}/c][\mathbf{v}/d][\mathbf{z}/e]\big(E_{1/2}\big)\Big) = E^a_{1/2}$

Theorem 5. *Emptiness checking for timed action shuffle expressions without renaming or for preemptive shuffle expressions with action semantics is undecidable.*

We end with the sketch of the proof that timed action shuffle expressions without intersection are more expressive than timed regular expressions. Our example is:

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

We also rely on the following observation (that is proved e.g. in [13, 8]) saying roughly that timed words having $n+1$ action symbols and whose time intervals belong to the same $(n+1)$-dimensional region cannot be distinguished by timed automata:

Remark 3. For any timed regular expression E and $w = a_0t_1a_1 \ldots a_{n-1}t_na_n \in \|E\|$ if we have $w' = a_0t'_1a_1 \ldots a_{n-1}t'_na_n$ ($a_i \in \Sigma$, $t_i, t'_i \in \mathbb{R}_{\geq 0}$) such that for all $1 \leq i \leq j \leq n$,

$$\Big[\sum\nolimits_{i\leq k\leq j} t_k\Big] = \Big[\sum\nolimits_{i\leq k\leq j} t'_k\Big] \text{ and } \mathsf{frac}\Big(\sum\nolimits_{i\leq k\leq j} x_k\Big) = \mathsf{frac}\Big(\sum\nolimits_{i\leq k\leq j} x'_k)\Big) \quad (10)$$

then $w' \in \|E\|$. Here $[\alpha]$ is the integral part and $\mathsf{frac}(\alpha)$ is the fractional part of $\alpha \in \mathbb{R}$.

Hence, we may see that $w_1 = a\,0.5\,\hat{a}\,b\,0.5\,\hat{b}\,c\,0.5\,\hat{c} \in \|E_0\|$, $w_2 = a\,0.3\,\hat{a}\,b\,0.7\,\hat{b}\,c\,0.3\,\hat{c} \notin \|E_0\|$, and w_1 and w_2 meet the condition (10). Therefore no timed regular expression can be equivalent to E_0.

7 Conclusions and comments

We have seen that shuffle regular expressions with renaming are equivalent to stopwatch automata, and that, even without renaming, they still have an undecidable emptiness problem. The use of weakly monotonic time was essential in the proof of the last result. We do not know whether, in a strongly monotonic time setting, this theorem still holds.

Note that the automata encodings of [9] for the undecidability for stopwatch automata cannot be directly transformed into expressions without renamings or into preemptive scheduling expressions: the paper [9] utilizes a technique called "wrapping of clock values" which, when translated to shuffle expressions, require renamings.

An interesting direction of study concerns the expressive power of shuffle expressions w.r.t. preemptive scheduling problems, especially on defining of a class of expressions embodying *scheduling strategies*.

Finally, it is not very hard to figure out that shuffle regular expressions without intersection have a decidable emptiness problem. The the proof idea would proceed in a "compositional" manner along the following three observations:

- $\|E_1E_2\| \neq \emptyset$ iff $\|E_1\| \neq \emptyset$ and $\|E_1\| \neq \emptyset$, and similarly for $E_1 \sqcup\!\sqcup E_2$.
- $\|E_1 + E_2\| \neq \emptyset$ iff $\|E_1\| \neq \emptyset$ or $\|E_1\| \neq \emptyset$.
- $\|E\| \neq \emptyset$ iff $\|E^*\| \neq \emptyset$.

References

1. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
2. E. Asarin. Challenges in timed languages. *Bulletin of EATCS*, 83, 2004.
3. E. Asarin, P. Caspi, and O. Maler. A Kleene theorem for timed automata. In *Proceedings of LICS'97*, pages 160–171, 1997.
4. E. Asarin, P. Caspi, and O. Maler. Timed regular expressions. *Journal of ACM*, 49:172–206, 2002.
5. P. Bouyer and A. Petit. Decomposition and composition of timed automata. In *Proceedings of ICALP'99*, volume 1644 of *LNCS*, pages 210–219, 1999.
6. C. Dima. Kleene theorems for event-clock automata. In *Proceedings of FCT'99*, volume 1684 of *LNCS*, pages 215–225, 1999.
7. C. Dima. Real-time automata. *Journal of Automata, Languages and Combinatorics*, 6:3–23, 2001.
8. C. Dima. A nonarchimedian discretization for timed languages. In *Proceedings of FORMATS'03*, volume 2791 of *LNCS*, pages 161–181, 2003.
9. T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata. *J. Comput. Syst. Sci*, 57:94–124, 1998.
10. P. Herrmann. Renaming is necessary in timed regular expressions. In *Proceedings of FST&TCS'99*, volume 1738 of *LNCS*, pages 47–59, 1999.
11. J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley/Narosa Publishing House, 1992.
12. P. Krcál and Wang Yi. Decidable and undecidable problems in schedulability analysis using timed automata. In *Proceedings of TACAS'04*, volume 2988 of *LNCS*, pages 236–250, 2004.
13. J. Ouaknine and James Worrell. Revisiting digitization, robustness, and decidability for timed automata. In IEEE Computer Society Press, editor, *Proceedings of LICS'03*, pages 198–207, 2003.