A machine-independent characterization of timed languages

Mikołaj Bojańczyk[⋆] and Sławomir Lasota[⋆⋆]

Institute of Informatics, University of Warsaw

Abstract. We use a variant of Fraenkel-Mostowski sets (known also as nominal sets) as a framework suitable for stating and proving the following two results on timed automata. The first result is a machine-independent characterization of languages of deterministic timed automata. As a second result we define a class of automata, called by us timed register automata, that extends timed automata and is effectively closed under minimization.

1 Introduction

This paper studies minimization of deterministic timed automata [2]. Existing approaches to this problem explicitly minimize various resources used by an automaton, such a locations or clocks, see [1, 8, 14–16]. We take a different approach, which abstracts away from the syntax of a timed automaton, and focuses on the recognized language, and specifically its Myhill-Nerode equivalence relation. Our notion of minimality is described by the following definition.

Definition 1. *An automaton for a language L is called* minimal *if for every two words w, w' the following conditions are equivalent:*

– *The words are equivalent with respect to Myhill-Nerode equivalence.*
– *The states reached after reading the words are equal.*

In the case of a deterministic timed automaton, the term "state" refers to the location (or control state) and the valuation of clocks. One of the main contributions of this paper is a minimization algorithm for deterministic timed automata. Of course in the case of timed automata, Myhill-Nerode equivalence has infinitely many equivalence classes, e.g. in the language

$$\{t_1 \cdots t_n \in \mathbb{R}^* : t_i = t_{i-1} + 1 \text{ for all } i \in \{2, \ldots, n\}\},$$

the equivalence class of a word is determined by its last letter.

A new automaton model. There is a technical problem with minimizing deterministic timed automata: the minimization process might leave the class of timed automata, as witnessed by the following example.

[⋆] Supported by the ERC Starting Grant "Sosna".
[⋆⋆] Supported by the FET-Open grant agreement FOX, number FP7-ICT-233599.

Example 1. Consider the following language $L \subseteq \mathbb{R}^*$. A word belongs to L if and only if it has exactly three letters $t_1, t_2, t_3 \in \mathbb{R}$, and the following conditions hold.

- The letter t_2 belongs to the open interval $(t_1; t_1 + 2)$;
- The letter t_3 belongs to the open interval $(t_1 + 2; t_1 + 3)$;
- The letters t_2 and t_3 have the same fractional part, i.e. $t_3 - t_2 \in \mathbb{Z}$.

This language is recognized by a deterministic timed automaton. After reading the first two letters t_1 and t_2, the automaton stores t_1 and t_2 in its clocks. This automaton is not minimal in the sense of Definition 1. The reason is that the words $(0, 0.5)$ and $(0, 1.5)$ are equivalent with respect to Myhill-Nerode equivalence, but the automaton reaches two different states. Any other timed automaton would also reach different states, as timed automata may reset clocks only on time-stamps seen in the input word (unless ε-transitions are allowed).

Because of the example above, we need a new definition of automata. We propose a straightforward modification of timed automata, which we call *timed register automata.* Roughly speaking, a timed register automaton works like a timed automaton, but it can modify its clocks, e.g. increment or decrement them by integers[1]. For instance, in language L from Example 1, the minimal automaton stores not the actual letter t_2, but the unique number in the interval $(t_1; t_1 + 1)$ that has the same fractional part as t_2.

We prove that timed register automata can be effectively minimized.

Typically, minimization corresponds to optimization of resources of an automaton. In case of timed automata, the resources seem to be locations and clocks, but maybe also constants used in the guards, anything else? One substantial novelty of our approach is that the kind of resource we optimize is not chosen ad hoc, but derived directly from Myhill-Nerode equivalence. Myhill-Nerode equivalence is an abstract concept; and therefore we need a tool that is well-suited to abstract concepts. The tool we use is *Fraenkel-Mostowski sets.*

Fraenkel-Mostowski sets. By these we mean a set theory different from the standard one, originating in the work of Fraenkel and Mostowski (see [10] for the references), and thus called by us *Fraenkel-Mostowski sets* (FM sets in short). Much later a special case of this set theory has been rediscovered by Gabbay and Pitts [11, 10] in the semantics community, as a convenient way of describing binding of variable names. Motivated by this important application, Gabbay and Pitts use the name *nominal sets* for the special case of FM sets they consider. Finally, FM sets (under the name "nominal G-sets") have been used in [3] to minimize automata over infinite alphabets, such as Francez-Kaminski finite-memory automata [9]. The paper [3] is the direct predecessor of the present paper.

In the setting of [3] (see also the full version [4]), FM sets are parametrized by a *data symmetry*, consisting of a set of data values together with a group G of permutations of this set. For instance, finite-memory automata are suitably represented in the data symmetry of all permutations of data values. To

[1] A certain restriction to the model is required to avoid capturing Minsky machines.

model timed automata, and even timed register automata, we choose the *timed symmetry*, based on the group of automorphisms of the structure[2]

$$(\mathbb{R}, <, +1).$$

Despite that this data symmetry presents several technical challenges, we show that FM sets can be used to solve nontrivial algorithmic problems, such as the minimization problem. A more accurate description of this paper is that we study automata in FM sets under the timed symmetry; and these automata happen to capture timed automata, and even timed register automata. In particular, we study languages where the timestamps appearing in a word are not necessarily increasing.

The second principal contribution of this paper is an exact characterization of the languages recognized by deterministic timed automata. The characterization is in the style of the Myhill-Nerode theorem, and is machine-independent, in the sense that it does not refer to any notion of recognizing device.

Summary of contributions. Below are the main contributions of our paper.

1. We introduce a new class of automata, called timed register automata, which generalize timed automata.
2. We prove that, unlike for deterministic timed automata, deterministic timed register automata are closed under minimization. We also give a minimization algorithm for timed register automata (Theorem 3 in Section 2).
3. We study automata in Fraenkel-Mostowski sets, under the timed symmetry.
4. We prove a kind of Myhill-Nerode theorem, which characterizes exactly the languages of deterministic timed automata (Theorem 5 in Section 4).

Related research. We only mention here a few related papers we are aware of. Minimization of (nondeterminstic) timed automata has been studied in particular in [1, 14, 16], with respect to bisimulation equivalence. As we mention later, our approach extends easily to bisimulation. On the negative side, minimization of nondeterministic automata with respect to language equivalence is undecidable, cf. [15, 8]. A characterization of deterministic timed languages using finite monoids has been proposed in [6]. Our characterization is of a different nature, being based on *orbit-finiteness* of the set of equivalence classes of Myhill-Nerode equivalence. Another machine-independent characterization of deterministic timed languages has been given in [13].

2 Timed register automata

In this paper, we study timed automata as a special case of automata where the alphabet is of the form $A \times \mathbb{R}$, where A is a finite set and $\mathbb{R}$ is the real numbers. In a letter $(a, t) \in A \times \mathbb{R}$, we call a the label and t the timestamp. Timed automata accept only words where the timestamps increase from left to

[2] Studying this group has been suggested to us by James Worrell.

right, call such words *monotonic*. Unlike timed automata, some of the automata we study in this paper can accept non-monotonic words.

Constraints. A *constraint* over variables $x_1, \dots, x_n$ is any quantifier free formula that uses the variables, the binary predicate $\leq$, and the unary function $+1$. Examples of constraints include

$$x \leq (y+1)+1 \quad \wedge \quad (y+1)+1 \leq x.$$

When writing constraints, we sometimes use syntactic sugar, for instance writing the above constraint as $x = y+2$. A constraint over variables $x_1, \dots, x_n$ defines a subset $X \subseteq \mathbb{R}^n$.

A constraint φ is called *maximal* if every other constraint on the same variables is either implied by φ, or inconsistent with φ. An example of a maximal constraint is

$$x_2 = x_1 + 1 \quad \wedge \quad x_2 < x_3 < x_2 + 1.$$

The constraint $x < y < x+2$ is not maximal, since it is independent with $y < x+1$. Not every constraint is equivalent to a finite disjunction of maximal constraints, for instance the constraint $x < y$.

Clearly, maximal constraints describe those *regions* that are bounded.

Timed register automata. We now define an automaton model, which can recognize languages over alphabets of the form $A \times \mathbb{R}$. A *(nondeterministic) timed register automaton* $\mathcal{A}$ is given by the following ingredients.

- A finite set A of *labels*.
- A finite set Loc of *locations*, also called *control states*.
- Subsets of the locations for the *initial* and *final* locations.
- For each location $l \in \text{Loc}$, a set X_l of register names[3].
- For every two locations $l, k \in \text{Loc}$, and every label $a \in A$, a constraint (not necessarily maximal) which defines a subset

$$\delta_{l,a,k} \subseteq \mathbb{R}^{X_l} \times \mathbb{R} \times \mathbb{R}^{X_k}.$$

We assume that every initial location has an empty set of register names.

A *state* of the automaton is defined to be a pair (l, η), where l is a location and η is a function, called the *register valuation*, of the form $\eta : X_l \to \mathbb{R}$. We write $Q_\mathcal{A}$ for the set of states of an automaton $\mathcal{A}$. This set is infinite if the automaton uses registers.

The semantics of the automaton is defined in the standard way. One defines the *transition relation*

$$\delta_\mathcal{A} \subseteq Q_\mathcal{A} \times (A \times \mathbb{R}) \times Q_\mathcal{A},$$

[3] A simplified version, where the set of register names does not depend on the location, would not minimize well. The reason is that the number of reals necessary to remember may depend on location. Ignoring some minor differences, the simplified version resembles updatable timed automata of [5].

to be the set of triples $(l, \eta), (a, t), (k, \mu)$ such that $(\eta, t, \mu) \in \delta_{l,a,k}$. A run over an input word from $(A \times \mathbb{R})^*$ is a sequence of states that starts in an initial state and is consistent with the transition relation.

A timed register automaton is called *deterministic* if there is one initial location and the transition relation $\delta_{\mathcal{A}}$ is a function $\delta_{\mathcal{A}} : Q_{\mathcal{A}} \times (A \times \mathbb{R}) \to Q_{\mathcal{A}}$.

Timed register automata, as defined above, are too powerful (a similar undecidability result is shown in [5]):

Theorem 1. *Emptiness is undecidable for deterministic timed register automata.*

Proof. By simulating a Minsky machine. The automaton has three register names: x, y, z. The idea is that z represents zero, $x - z$ is the value of the first counter and $y - z$ is the value of the second counter. Since the automaton can use the $+1$ in its transition relation, it can increment and decrement the counters. The zero tests are simulated by testing $x = z$ or $y = z$. $\square$

The reason why the undecidability proof above works is that we allow a state to store, at the same time, real numbers which are very far from each other. This motivates a restriction on timed register automata to be defined now.

Constrained timed register automata. In a *constrained timed register automaton*, for each location l there is a maximal constraint φ_l over the register names of l, called the *legality constraint.* In a constrained automaton, the notion of state is changed: a state (l, μ) must be such that the register valuation μ satisfies the constraint φ_l. Despite the different semantics, a constrained timed register automaton can be easily seen to be a special case of a timed register automaton, because legality constraints can be enforced by the transition relation.

The idea of adding legality constraints might seem an ugly fix. As we shall see later, constrained timed register automata have an elegant interpretation in terms of FM sets. Also, they are powerful enough to simulate timed automata.

Theorem 2. *Emptiness is decidable for constrained timed register automata.*

As our first main result, we state:

Theorem 3. *The class of constrained timed register automata is closed under minimization. There is an algorithm that computes, for a given constrained timed register automaton, the minimal automaton.*

Speaking abstractly, the minimal automaton is the syntactic automaton, or, in other words, the quotient of a given automaton by language equivalence; this will become apparent when in Section 3 we will observe that timed register automata are a subclass of automata in FM sets under the timed symmetry. Speaking concretely, we minimize the number of locations, and the number of register variables in each location.

Our minimization algorithm adopts the classical idea of iterative partition refinement, and works equally well for bisimulation of nondeterministic automata.

Timed automata. Timed automata [2] are defined similarly as timed register automata above. A timed automaton has a number of clock variables, that may be used to store the current timestamp and to compare it against timestamps read later on. The transition relation of a timed automaton is described using a subset of constraints, in the sense of the above definition. With these respects, timed automata seem to be a subclass of constrained timed register automata.

Timed automata have however one additional feature, not reflected in our definitions above: the clock variables are initially set to 0. In consequence, only non-negative timestamps are considered. Intuitively, a timed automaton is aware of the time that has elapsed from some *absolute* moment 0, while our automata are only aware of the *relative* time separating timestamps in the input. In particular, languages recognized by timed register automata are always closed under translations, i.e., for any $d \in \mathbb{R}$, the permutation $x \mapsto x + t$ preserves L:

$$L + t = L.$$

A language $L \subseteq (A \times \mathbb{R}^{\geq 0})^*$ can be encoded as the following language closed under translations, which has essentially the same structure as L:

$$\overrightarrow{L} \ = \ \bigcup_{\substack{t \in \mathbb{R} \\ a \in A}} ((a, 0)\ L) + t \ = \ \bigcup_{\substack{t \in \mathbb{R} \\ a \in A}} (a, t)\ (L + t) \ \subseteq \ (A \times \mathbb{R})^*.$$

Thus, in this paper we only consider languages that are closed under translations. On the level of timed automata, this property may be enforced by assuming that all the clock variables are *uninitialized* (that is, initially undefined), similarly like in finite memory automata of Francez and Kaminski [9].

Theorem 4. *For every (deterministic) timed automaton with uninitialized clocks one can compute an equivalent (deterministic) constrained timed register automaton.*

The idea of the proof is to translate regions of a timed automaton to locations of a timed register automaton. Unbounded regions are eliminated by projecting onto bounded coordinates. One additional register checks monotonicity.

Constrained timed register automata are strictly more expressible than timed automata, as shown in the example below.

Example 2. Let A be a singleton, thus $A \times \mathbb{R}$ is essentially $\mathbb{R}$. The language

$$L = \{t_1 \dots t_n \ : \ n \geq 2,\ t_n - t_1 \in \mathbb{N},\ t_{i+1} - t_i \leq 1 \text{ for } i < n\}$$

is not recognized by a timed automaton, but is recognized by a deterministic constrained timed register automaton with two registers. The automaton stores initially t_1 in its register, and then increments its value, say t, by 1 at every input letter greater than t. It accepts whenever an input letter equals t.

Due to Theorem 4, the minimization algorithm of Theorem 3 works for deterministic timed automata as well. How does the definition of minimality from Definition 1 correspond to resources of a timed automaton? The most appropriate to say is that we minimize the number of regions, and the number of clocks in each region. Indeed, as regions of timed automata are translated to locations of timed register automata, each region may be optimized independently. We however honestly note that the number of locations of the minimal automaton may be greater than the number of locations of an original timed automaton.

3 Fraenkel-Mostowski sets and their automata

The definition of Fraenkel-Mostowski sets (FM sets) is parametrized by a *data symmetry* $(\mathbb{D}, G)$, which consists of a set $\mathbb{D}$ of *data values* and a subgroup G of the group of all bijections of $\mathbb{D}$. Examples of data symmetries include:

- The *classical symmetry*, where the set of data values is empty, and the group has only the identity. FM sets in the classical symmetry are going to be normal sets.
- The *equality symmetry*, where the set of data values is a countably infinite set, and the group contains all bijections. FM sets in the equality symmetry are essentially the same thing as nominal sets in [11] or FM sets in [10].
- The *timed symmetry*, where the set of data values is the real numbers, and the group contains all permutations of real numbers that preserve the order relation $\leq$ and the successor function $x \mapsto x + 1$ (we call such permutations *timed permutations*). This is the data symmetry that we use in this paper.

Intuitively speaking, normal sets are built out of empty sets and brackets $\{$ and $\}$. The intuition behind FM sets is that they can in addition use data values as atomic elements. Our presentation below is motivated by [10].

Fix a data symmetry $(\mathbb{D}, G)$. Consider first the *cumulative hierarchy* of sets with data values, which is a hierarchy of sets indexed by ordinal numbers and defined as follows. The empty set is the unique set of rank 0. A set of rank α is any set whose elements are sets of rank smaller than α, or data values. A permutation $\pi \in G$ can be applied to a set X in the hierarchy, by renaming the data values belonging to X, and the data values belonging to elements of X, and so on. The resulting set, which has the same rank, is denoted by $X \cdot \pi$.

A set C of data values is said to be a *support* of a set X in the cumulative hierarchy if $X \cdot \pi = X \cdot \sigma$ holds for every permutations $\pi, \sigma \in G$ which agree on elements of C. A set is called *finitely supported* if it has some finite support. We use the name *FM set* for a set in the cumulative hierarchy which is hereditarily finitely supported, which means that it is finitely supported, the sets belonging to it are finitely supported, and so on.

The support of an FM set is not unique, e.g. supports are closed under adding data values. A set with empty support is called *equivariant*.

Example 3. An example of an equivariant FM set in the timed symmetry is $\mathbb{R}$ itself. Another example is $\mathbb{R}^*$. A tuple $(x_1, \ldots, x_n) \in \mathbb{R}^*$ is supported by the set $\{x_1, \ldots, x_n\}$. The set $\mathbb{R} - \{0\}$ is not equivariant; it is supported by $\{0\}$.

For some data symmetries, including the classical and equality ones, one can show that every FM set has the least support. However, FM sets in the timed symmetry do not have least supports. For instance, the set $\mathbb{R} - \{0\}$ is not supported by the empty set, but it is supported by the sets $\{0\}$ or $\{1\}$. This is because if π is a timed permutation, then $\pi(1) = 1$ is equivalent to $\pi(0) = 0$.

In many respects, FM sets behave like normal sets. For instance, if X, Y are FM sets, then $X \times Y$, $X \cup Y$, X^* and the finite powerset of X are all FM sets. Another example is the family of subsets of X that have finite supports. The appropriate notion of a function between FM sets X and Y is that of a *finitely supported function*, which is a function from X to Y whose graph is an FM set.

Orbit-finite FM sets. From our perspective, the key property of FM sets is their more relaxed notion of finiteness. Suppose that X is an FM set. For a set of data values C, define the C-orbit of an element $x \in X$ to be the set

$$\{x \cdot \pi : \pi \in G \text{ and } \pi \text{ is the identity on } C\}.$$

If C supports X, then the C-orbits form a partition of X. The set X is called orbit-finite if the partition into C-orbits has finitely many parts, for some C which supports X. Observe that the number of C-orbits increases as the set C grows. Therefore, a set is orbit-finite if it has a finite number of orbits for some minimal set C that supports it. In particular, an equivariant set is orbit-finite if and only if it has finitely many $\emptyset$-orbits.

In many data symmetries orbit-finite sets are closed under product, but not in the timed symmetry as illustrated in Example 4.

Example 4. The set $\mathbb{R}$ is orbit-finite, namely it has one $\emptyset$-orbit. The set $\mathbb{R}^2$ is not orbit-finite. The $\emptyset$-orbits are of the following form:

$$\{(x,y) : x - y = k\} \qquad \{(x,y) : x - y \in (k, k+1)\} \qquad \text{for all } k \in \mathbb{Z}.$$

Observe that two orbits of the first kind, say $\{(x,y) : x - y = k\}$ and $\{(x,y) : x - y = l\}$, are equivariantly isomorphic via the mapping $(x,y) \mapsto (x, y + (k - l))$. Likewise, every two orbits of the second kind are mutually isomorphic. Another example of two isomorphic but distinct orbits in $\mathbb{R}^*$ is $\mathbb{R}$ and $\{(x,x,x) : x \in \mathbb{R}\}$. There are infinitely many equivariant isomorphisms between these two orbits, including $x \mapsto (x,x,x)$ and $x \mapsto (x+1, x+1, x+1)$.

Automata. The definition of automata in FM sets is exactly like the definition of automata in normal sets, except that the notion of finiteness is relaxed to orbit-finiteness. Specifically, a *nondeterministic FM automaton* is a tuple

$$(A, Q, I, F, \delta) \qquad I, F \subseteq Q \qquad \delta \subseteq Q \times A \times Q$$

where the alphabet A, states Q, initial states $I \subseteq Q$, final states $F \subseteq Q$ and transitions $\delta \subseteq Q \times A \times Q$ are FM sets, and all of them except for δ are required to be orbit-finite. (We come back to the orbit-finiteness of δ in Example 5.) The definition of acceptance is as usual for automata. An automaton is called *equivariant* if all of its components are equivariant. From now on, we only study equivariant automata.

Example 5. Consider the language $L \subseteq \mathbb{R}^*$ which contains words where some letter appears twice. This language is recognized by a nondeterministic FM automaton whose states are: an initial state q, one state q_x for each real number x, and a single accepting state $\top$. The transition relation contains triples

$$(q, x, q) \qquad (q, x, q_x) \qquad (q_x, y, q_x) \qquad (q_x, x, \top) \qquad (\top, x, \top)$$

for every real numbers x, y. The transition relation is not orbit-finite, because the set of transitions (q_x, y, q_x) is isomorphic to $\mathbb{R}^2$. In general, the transition relation will necessarily have infinitely many orbits in any automaton which stores real numbers in its state, and which reads arbitrary input letters.

A *deterministic FM automaton* is the special case of a nondeterministic one, where the transition relation is a function $\delta : Q \times A \to Q$, and where the set of initial states contains only one state. From now on, we only study equivariant deterministic automata and work exclusively in the timed symmetry.

Comparing the models. So far, we have introduced two kinds of automata. In Section 2, we have introduced timed register automata, and we have identified a subclass of constrained timed register automata. In Section 3, we have introduced automata in FM sets. In this section, we show that in the specific case of FM sets in the timed symmetry, the two kinds of automata are closely related. We only study the deterministic case, but the nondeterministic case is analogous. The results are summed up in Figure 1.

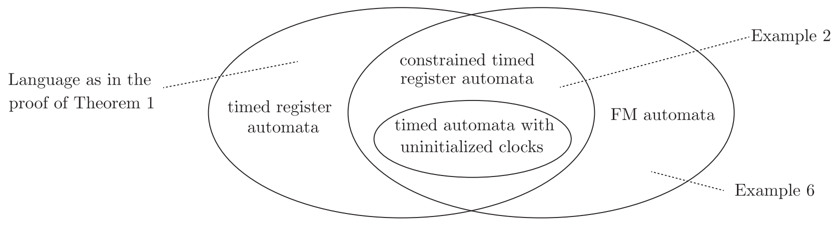
Language as in the proof of Theorem 1
timed register automata
constrained timed register automata
timed automata with uninitialized clocks
FM automata
Example 2
Example 6

Fig. 1. Timed register automata, and FM automata in the timed symmetry.

We first show that a deterministic timed register automaton is almost a special case of a deterministic FM automaton. The input alphabet, which is a set of the form $A \times \mathbb{R}$, for a finite set A is an equivariant orbit-finite FM set. The number of orbits is the size of A, because permutations of data values (= timestamps) do not change the labels. Recall that a state of a timed register automaton consists of a location and a valuation of registers. Thus the set of all states is an equivariant FM set, since it is basically a set of tuples of real numbers. In the same way, the initial and accepting states are equivariant subsets, because they are identified by their locations, and locations are not changed by permutations of data values. Finally, transition function of a timed register automaton is equivariant, because it is defined in terms of the order and successor, both preserved by timed permutations.

So why is a deterministic timed register automaton not necessarily an FM automaton? Because the states are not, in general, an orbit-finite FM set. For instance, if an automaton has two registers in some location, then its states will not be orbit-finite for the same reason as $\mathbb{R}^2$. This is where the constraints on the register valuations, as defined in Section 2, come in. The following lemma shows that maximal constraints can be used to enforce an orbit-finite state space.

Lemma 1. *The following conditions are equivalent for a subset $X \subseteq \mathbb{R}^n$:*

– *X is equivariant and has one orbit.*
– *X is defined by a maximal constraint.*

As a conclusion, a constrained timed register automaton is exactly the same thing as a timed register automaton, whose state space is orbit-finite.

So far we have shown that constrained timed register automata are included in FM automata. The inclusion is strict, as the transition function in a timed register automaton is defined by constraints, while in the abstract definition, the transition function is only required to be equivariant. Not all equivariant transition functions are definable by constraints, as shown in the following example.

Example 6. Suppose that $K \subseteq \mathbb{Z}$ is any set of integers, e.g. the prime numbers. Consider the language $\mathrm{diff}(K) = \{t_1\, t_2 \in \mathbb{R}^2 \ : \ t_2 - t_1 \in K\}$. Regardless of K, this language can be recognized by a deterministic FM automaton. The state space of the automaton has four orbits: an initial state ϵ, an accepting state $\top$, a rejecting sink state $\bot$, and one state q_t for every real number t. The automaton starts in the initial state ϵ. The transition function is:

$$\delta(\epsilon, t) = q_t \qquad \delta(\bot, t) = \bot \qquad \delta(q_s, t) = \begin{cases} \top & \text{if } t - s \in K \\ \bot & \text{otherwise} \end{cases} \qquad \delta(\top, t) = \top$$

The transition function is easily seen to be equivariant. For most K, however, it is not defined by a constraint (one argument is that there are uncountably many choices for K, and only countably many choices for a constraint).

Example 6 implies that the abstract definition of a deterministic FM automaton is too powerful. For instance, arbitrary FM automata cannot be represented in a finite way. Restricting equivariant functions to those definable by constraints makes the automata manageable, but it is not necessarily the only solution to the problem. We do not investigate other solutions in this paper.

4 Characterization of deterministic timed automata

In this section we provide a machine-independent characterization of the class of languages recognized by deterministic timed automata.

Every language recognized by a deterministic timed automaton with uninitialized clocks is equivariant and contains only monotonic words. Finally, the set of equivalence classes of Myhill-Nerode equivalence is orbit-finite. As shown in Example 6, these conditions are not sufficient even to characterize nondeterministic orbit-finite timed register automata. One additionally needs to say, roughly, that only recent timestamps can be remembered, and older timestamps must be forgotten. Our formulation of this condition is as follows.

For two nonempty words $u, w \in (A \times \mathbb{R})^+$ (think of monotonic words) and $M \in \mathbb{N}$ we write $u <_M w$ to mean that the first timestamp in w is larger than the last timestamp in u, by at least M.

Definition 2. *Let $M \in \mathbb{N}$. A language $L \subseteq (A \times \mathbb{R})^*$ is called* M-forgetful *if for every words $u, w \in (A \times \mathbb{R})^+$, $v \in (A \times \mathbb{R})^*$ and a timed permutation π such that $v \cdot \pi = v$, $u <_M w$ and $u \cdot \pi <_M w$, it holds:*

$$u\,v\,w \in L \;\Leftrightarrow\; (u \cdot \pi)\,v\,w \in L. \tag{1}$$

Observe that M-forgetfulness implies M'-forgetfulness for all $M' > M$. Note that $v \cdot \pi = v$ implies $(u\,v) \cdot \pi = (u \cdot \pi)\,v$ and that if L is equivariant then the property (1) may be equivalently written as $u\,v\,w \in L \;\Leftrightarrow\; u\,v\,(w \cdot \pi) \in L$.

Example 7. The language L from Example 2 in Section 2 is not M-forgetful for any $M \geq 0$. Indeed, instantiating Definition 2 with

$$u = 0.4 \qquad v = 1.2\ 2.2\ \ldots\ M{+}0.2\ M{+}1.2 \qquad w = M{+}1.4$$

and any timed permutation π satisfying $\pi(0.4) = 0.3$ and $\pi(0.2) = 0.2$, we get a contradiction, as $0.4\ v\ w \in L$ while $0.3\ v\ w \notin L$.

Example 8. The language of all monotonic words is 0-forgetful. The language "for some timestamp t, both t and $t + 3$ appear in the word" is 3-forgetful but not 2-forgetful.

Theorem 5. *Let A be a finite set of labels. For a language $L \subseteq (A \times \mathbb{R})^*$, the following conditions are equivalent:*

- *L is recognized by a deterministic timed automaton with uninitialized clocks.*
- *L satisfies simultaneously the following conditions:*
  1. *L is equivariant;*
  2. *L contains only monotonic words;*
  3. *L is M-forgetful for some threshold $M > 0$; and*
  4. *the set of equivalence classes of the Myhill-Nerode equivalence $\sim_L$ is orbit-finite.*

Note that the set of equivalence classes of $\sim_L$ is an (equivariant) FM set when L is an (equivariant) FM set. Even in presence of condition 3, condition 4 is still necessary, as shown by the following example.

Example 9. Consider the language containing all monotonic timed words of the form $t_1\,t_2\,\ldots\,t_n\,(t_1{+}1)\,(t_2{+}1)\,\ldots\,(t_n{+}1)$, for $n \geq 0$. The language is 1-forgetful, but its syntactic automaton is orbit-infinite.

5 Future work

Our approach based on Fraenkel-Mostowski sets may be further elaborated.

We consider a subclass of orbit-finite automata where the transition function (or relation) is definable by constraints. These restrictions are sufficient to capture timed automata, but there may be other manageable restrictions that are more liberal. As a natural continuation of this work we plan to pursue automata with semi-linear transition functions. We suppose that one would be able to capture in this framework, among the others, periodic time constraints, cf. [7], or some subclasses of hybrid automata, like linear hybrid automata [12].

Another urgent challenge is to relate our approach to the previous work, in particular to minimization of [1, 14, 16] and to characterizations of [6] and [13].

Acknowledgments. We kindly thank anonymous reviewers for insightful comments and valuable suggestions.

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, D. L. Dill, and H. Wong-Toi. Minimization of timed transition systems. In *CONCUR*, pages 340–354, 1992.
2. R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
3. M. Bojańczyk, B. Klin, and S. Lasota. Automata with group actions. In *Proc. LICS'11*, pages 355–364, 2011.
4. M. Bojańczyk, B. Klin, and S. Lasota. Automata theory in nominal sets. 2012. Submitted. Accessible at `http://www.mimuw.edu.pl/~sl/PAPERS/lics11full.pdf`.
5. P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Updatable timed automata. *Theor. Comput. Sci.*, 321(2-3):291–345, 2004.
6. P. Bouyer, A. Petit, and D. Thérien. An algebraic approach to data languages and timed languages. *Inf. Comput.*, 182(2):137–162, 2003.
7. C. Choffrut and M. Goldwurm. Timed automata with periodic clock constraints. *Journal of Automata, Languages and Combinatorics*, 5(4):371–404, 2000.
8. O. Finkel. Undecidable problems about timed automata. *CoRR*, abs/0712.1363, 2007.
9. N. Francez and M. Kaminski. Finite-memory automata. *TCS*, 134(2):329–363, 1994.
10. M. Gabbay. Foundations of nominal techniques: logic and semantics of variables in abstract syntax. *Bulletin of Symbolic Logic*, 17(2):161–229, 2011.
11. M. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Asp. Comput.*, 13(3-5):341–363, 2002.
12. T. A. Henzinger. The theory of hybrid automata. In *LICS*, pages 278–292, 1996.
13. Oded Maler and Amir Pnueli. On recognizable timed languages. In *FoSSaCS*, pages 348–362, 2004.
14. J. Springintveld and F. W. Vaandrager. Minimizable timed automata. In *FTRTFT*, pages 130–147, 1996.
15. S. Tripakis. Folk theorems on the determinization and minimization of timed automata. *Inf. Process. Lett.*, 99(6):222–226, 2006.
16. M. Yannakakis and D. Lee. An efficient algorithm for minimizing real-time transition systems. *Formal Methods in System Design*, 11(2):113–136, 1997.