

Relating timed and register automata

Diego Figueira, Piotr Hofman, Slawomir Lasota

▶ **To cite this version:**

Diego Figueira, Piotr Hofman, Slawomir Lasota. Relating timed and register automata. Mathematical Structures in Computer Science, 2016, 26 (06), pp.993-1021. 10.1017/S0960129514000322 . hal-01686366

HAL Id: hal-01686366
https://hal.science/hal-01686366

Submitted on 17 Jan 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Under consideration for publication in Math. Struct. in Comp. Science

Relating timed and register automata[†]

DIEGO FIGUEIRA[1,2,3‡], PIOTR HOFMAN[3] and SŁAWOMIR LASOTA[3]

[1] *University of Edinburgh, 10 Crichton Street, Edinburgh EH8 9AB, UK*
[2] *INRIA, ENS Cachan, LSV, 61 avenue du Président Wilson 94235 Cachan, France*
[3] *Institute of Informatics, University of Warsaw, Banacha 2, 02-097 Warszawa, Poland*

Received 23 March 2011; Revised 20 September 2012

Timed automata and register automata are well-known models of computation over timed and data words respectively. The former has *clocks* that allow to test the lapse of time between two events, whilst the latter includes *registers* that can store data values for later comparison. Although these two models behave in appearance differently, several decision problems have the same (un)decidability and complexity results for both models. As a prominent example, emptiness is decidable for alternating automata with one clock or register, both with non-primitive recursive complexity. This is not by chance. This work confirms that there is indeed a tight relationship between the two models. We show that a run of a timed automaton can be simulated by a register automaton over ordered data domain, and conversely that a run of a register automaton can be simulated by a timed automaton. These are exponential time reductions hold both in the finite and infinite words settings. Our results allow to transfer decidability results back and forth between these two kinds of models, as well complexity results modulo an exponential time reduction. We justify the usefulness of these reductions by obtaining new results on register automata.

1. Introduction

Timed automata (Alur and Dill, 1994) and register automata (known originally as *finite-memory automata*) (Kaminski and Francez, 1994) are two widely studied models of computation. Both models extend finite automata with a kind of storage: *clocks* in the case of timed automata, capable of measuring the amount of time elapsed from the moment they were reset; and *registers* in the case of register automata, capable of storing a data value for future comparison. In this paper we are interested in decidability and complexity of standard decision problems for both models of automata. In particular,

[†] Authors emails: `dfigueir@inf.ed.ac.uk`, `sl@mimuw.edu.pl`, `ph209519@mimuw.edu.pl`.
An extended abstract of this work appeared as (Figueira, Hofman, and Lasota, 2010).
The first author acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under the FET-Open grant agreement FOX, number FP7-ICT-233599. The second and third authors acknowledge the financial support of the Polish Ministry of Science grant nr N N206 567840.
[‡] Corresponding author.

we focus on the problems of *nonemptiness* (Does an automaton $\mathcal{A}$ accept some word?), *universality* (Does an automaton $\mathcal{A}$ accept all words?), and *inclusion* (Are all words accepted by an automaton $\mathcal{A}$ also accepted by an automaton $\mathcal{B}$?).

The emptiness problem for nondeterministic timed or register automata is PSPACE-complete (Alur and Dill, 1994; Demri and Lazić, 2009). It becomes undecidable for *alternating* automata of both kinds (Lasota and Walukiewicz, 2005; Ouaknine and Worrell, 2005; Demri and Lazić, 2009), as soon as they have at least two clocks or registers (Alur and Dill, 1994; Demri and Lazić, 2009). Even the universality problem was shown undecidable for nondeterministic timed and register automata, respectively, with two clocks or registers (Alur and Dill, 1994; Neven et al., 2004; Demri and Lazić, 2009). A break-through result of (Ouaknine and Worrell, 2004) showed that universality becomes decidable for one clock timed automata. Later, the emptiness problem for one clock alternating timed automata was shown decidable. However, the computational complexity of this problem has been found to be non-primitive recursive (Lasota and Walukiewicz, 2005; Ouaknine and Worrell, 2005). Analogous (independent) results appeared for the other model: emptiness is decidable and non-primitive recursive for one register alternating automata (Demri and Lazić, 2009). For infinite words, both one clock and one register alternating automata are undecidable, as well as the universality problem of nondeterministic one clock/register automata (Lasota and Walukiewicz, 2008; Abdulla et al., 2005; Demri and Lazić, 2009). The analogies between the two models appear to some extent also at the level of proof methods. The decidability proofs for one clock/register alternating automata are based on similar well-structured transition systems; and both non-primitive recursive lower bounds are obtained by simulation of a kind of lossy model of computation. All these analogies between the two models rise a natural question about the relationship between them. This paper is an attempt to answer this question.[†]

Register automata were traditionally investigated over an unordered data domain. However, our model works on a data domain equipped with a total order. This is a necessary extension, that allows to simulate runs of timed automata, and to have a tight equivalence between the timed and the register models. Roughly speaking, the main contribution of this paper is to show that timed automata and register automata over an ordered data domain are equivalent models, as far as one concerns complexity and decidability of decision problems.

On a more technical level, we show that a run of a timed/register automaton on a timed/data word w may be simulated by a run of a register/timed automaton over a specially instrumented transformation of w, that we call *braid*. The reductions we exhibit are performed in exponential time, and keep the number of clocks equal to the number of registers, and preserve the mode of computation (alternating, nondeterministic, deterministic). Additionally, we show that the complement of all braids is recognizable by a nondeterministic one clock/register automaton. These results lead straightforwardly

[†] Another evidence of analogies between clock and register automata follows when one sees these automata classes from the perspective of sets with atoms, see Bojańczyk et al. (2011) and Bojańczyk and Lasota (2012). Yet another approach that applies to both kinds of automata is the algebraic characterization of Bouyer et al. (2003).

to reductions from decision problems for one class of automata to analogous problems for the other class, thus allowing us to carry over (un)decidability results and derive complexity bounds in both directions. These results are also extended to the case of ω-words.

As an application, our simulations allow to obtain known results on timed (or register) models as simple consequences of results on register (or timed) models. These include, e.g., that over finite words the emptiness problem of alternating 1 register automata is decidable (Demri and Lazić, 2009). In fact, our reductions yield decidability of the model extended with a total order over the data domain. As further examples of application, we show how the following complexity and decidability results for timed automata can be transferred to the class of register automata (the first two problem are for finite words, and the last two for ω-words):

— decidability of the inclusion problem between a nondeterministic (many clocks) automaton and an alternating 1-clock automaton (shown in (Lasota and Walukiewicz, 2008));
— decidability of the emptiness problem for an alternating (many clocks) automaton over a *bounded* time domain (shown in (Jenkins et al., 2010));
— decidability/undecidability of the non-Zeno emptiness problem for alternating 1-clock automata with weak acceptance conditions, shown in (Parys and Walukiewicz, 2009);
— non-elementary lower bound for the emptiness problem of safety 1-clock alternating automata, shown in (Bouyer et al., 2008).

In Sections 2– 5 we limit our study to *finite* timed and data words, as the first step in relating the timed and data settings. Then in Section 6 we show how to accommodate ω-words. Section 7 is devoted to applications.

2. Preliminaries

We fix a finite alphabet $\mathbb{A}$ for the sequel. We recall the definitions of alternating timed and register automata. To avoid inessential technical complications, we have deliberately chosen to formulate both the definitions in an analogous way. Our definitions are inspired by those in (Lasota and Walukiewicz, 2008; Ouaknine and Worrell, 2005; Demri and Lazić, 2009).

2.1. *Alternating register automata*

Fix an infinite data domain $\mathbb{D}$. *Data words* over $\mathbb{A}$ are finite sequences

$$w \;=\; (a_1, d_1)(a_2, d_2)\dots(a_n, d_n) \tag{1}$$

of pairs from $\mathbb{A} \times \mathbb{D}$. Additionally, assume a total order $\preceq$ over $\mathbb{D}$. The order may be chosen arbitrarily, and our results apply to all total orders. In particular, we do not assume that $\preceq$ is dense.

Ingredients of an *alternating register automaton* $\mathcal{A}$ over an alphabet $\mathbb{A}$ are: a finite set Q of states, partitioned into states owned by two players Eve and Adam; a distinguished

initial state; a finite set $\mathcal{R}$ of *register names* (or *registers* for short); and a finite set of transitions. Each transition is a triple $q \xrightarrow{o} p$, where q and p are states, and o is an *operation*, as described below.

We distinguish four kinds of operations: *label tests* 'test if the current label is a'; *data tests* 'test if the current datum $\sqsubseteq r$' (we write '$\sqsubseteq r$' for short), for $\sqsubseteq \in \{\prec, \preceq, \succ, \succeq\}$ and $r \in \mathcal{R}$; 'load current datum to r'; and 'go to next position'. E.g., '$\prec r$' checks if the current data value is strictly smaller than the value stored in register r. The equality '$= r$' and inequality '$\neq r$' tests may be easily simulated, as well as boolean combinations of data tests.

Register automata were typically defined till now over unordered data domain (with the exceptions, e.g., of (Benedikt et al., 2010) and (Bojańczyk et al., 2011)). For the purpose of relating the existing models, distinguish a subclass of register automata that only use equality '$= r$' and inequality '$\neq r$' data tests; we call them *order-blind automata*. Order-blind automata correspond to the model defined in (Demri and Lazić, 2009).

For a given input word w as in (1), a configuration of an automaton $\mathcal{A}$ is a triple (q, v, i) where q is a state, $v : \mathcal{R} \to \mathbb{D}$ is a valuation of registers and $i \in \{1 \dots n\}$ is a position in w. A transition $q \xrightarrow{o} p$ induces naturally a relation on configurations as defined in Table 2.1.

operation o	condition on $(q, v, i) \xrightarrow{o} (p, u, j)$
test if the current label is a	$a_i = a$ and $(u, j) = (v, i)$
test if the current datum $\sqsubseteq r$	$d_i \sqsubseteq v(r)$ and $(u, j) = (v, i)$
load current datum to r	$(u, j) = (v[r := d_i], i)$
go to next position	$i < n$ and $(u, j) = (v, i + 1)$

Table 1. Transitions between configurations of an alternating register automaton.

It is easy to simulate a transition $q \longrightarrow p$ with no operation that imposes no condition on configuration.

The data word w is accepted or not by $\mathcal{A}$ depending on the winner in the *acceptance game* played by Eve and Adam. The game begins in the *initial configuration* that consists of the initial state, the first position in w, and the valuation that assigns d_1 to every register (in this way we avoid undefined values in registers). In a configuration (q, v, i) the player that owns state q (we say that the configuration is owned by that player) chooses the next configuration (p, u, j) such that $(q, v, i) \xrightarrow{o} (p, u, j)$ for some transition $q \xrightarrow{o} p$. Eve wins if a configuration owned by Adam is reached from which no move can be done; otherwise Adam wins. The latter case includes infinite plays that may arise if the players stay in a position of w forever. Note that we do not need accepting states, as they may be simulated by a state owned by Adam with no outgoing transitions. Also, note that we do not require to read the whole word, the game may end before reading the last position.

The automaton $\mathcal{A}$ *accepts* w iff Eve has a winning strategy in the acceptance game. By $\mathcal{L}(\mathcal{A})$ denote the language of all data words accepted by $\mathcal{A}$.

2.2. *Alternating timed automata*

By a *timed word* over $\mathbb{A}$ we mean a finite sequence

$$w \;=\; (a_1, t_1)\,(a_2, t_2)\,\ldots\,(a_n, t_n) \tag{2}$$

of pairs from $\mathbb{A} \times \mathbb{R}_+$, with $t_1 < t_2 < \ldots < t_n$, where $\mathbb{R}_+$ is the set of non-negative reals. Each *time stamp* t_i denotes the amount of time elapsed since the beginning of the word. For simplicity, we prefer to work with strictly monotonic timed words, although the analogous results would hold for weakly monotonic words as well.

In an intentional analogy to register automata, ingredients of an *alternating timed automaton* $\mathcal{A}$ over an alphabet $\mathbb{A}$ are: a finite set Q of states, partitioned into states owned by Eve and Adam; a distinguished initial state; a finite set $\mathcal{C}$ of *clock names* (or *clocks* for short); and a finite set of transitions $q \xrightarrow{o} p$, where q and p are states, and o is an operation, as described below.

There are four kinds of operations: 'test if the current label is a'; the so called *clock constraint* 'test if $c \sqsubseteq k$' (we write '$c \sqsubseteq k$' for short), for $\sqsubseteq \in \{<, \leq, >, \geq\}$, $c \in \mathcal{C}$ and $k \in \mathbb{N}$; 'reset clock c'; and 'go to next position'. As before, boolean combinations of clock constraints can be easily simulated.

For a given input word w as in (2), a configuration of an automaton $\mathcal{A}$ is a triple (q, v, i) where q is a state, $v : \mathcal{C} \to \mathbb{R}_+$ is a valuation of clocks and i is a position in w. A transition $q \xrightarrow{o} p$ induces naturally a relation on configurations:

operation o	condition on $(q, v, i) \xrightarrow{o} (p, u, j)$
test if the current label is a	$a_i = a$ and $(u, j) = (v, i)$
test if $c \sqsubseteq k$	$v(c) \sqsubseteq k$ and $(u, j) = (v, i)$
reset clock c	$(u, j) = (v[c := 0], i)$
go to next position	$i < n$ and $(u, j) = (v{+}(t_{i+1} - t_i), i + 1)$

Table 2. Transitions between configurations of an alternating timed automaton.

A valuation $v{+}t$, for $t \in \mathbb{R}_+$, is obtained from v by increasing the values of all clocks by t.

Slightly overloading the notation, by $\mathcal{L}(\mathcal{A})$ we denote the language of all timed words accepted by $\mathcal{A}$, i.e., those words w for which Eve has a winning strategy in the *acceptance game*. The game begins in the *initial configuration* that consists of the initial state, the first position in w, and the valuation that assigns 0 to every clock. In a configuration (q, v, i) the player that owns state q chooses the next configuration (p, u, j) such that $(q, v, i) \xrightarrow{o} (p, u, j)$ for some transition $q \xrightarrow{o} p$. The winner is established similarly as for register automata.

2.3. *Modes of computation*

For both timed and register automata, we distinguish a subclass of nondeterministic automata. An automaton is nondeterministic if every configuration (q, v, i) owned by Adam has *at most one* outgoing transition $(q, v, i) \xrightarrow{o} (p, u, j)$. (Note that we can not

require that all states are owned by Eve as there are no accepting states.) Symmetrically, one may define co-nondeterministic automata by exchanging the roles of Adam and Eve. Deterministic automata are those that are simultaneously nondeterministic and co-nondeterministic; thus there is at most one allowed transition from every configuration and hence the ownership of states is relevant uniquely with respect to acceptance. The term alternating automata refers then to the full, unrestricted class.

Our notion of automata is slightly unusual, as we do not distinguish explicitly accepting states. As an illustration, consider a deterministic automaton and a state q with a transition (q, o, p), where the operation o is 'go to next position'. If state q is owned by Adam, it is Eve to win in state q if the current position in the word is the last position; thus q may be thought as an accepting state. Symmetrically, if state q is owned by Eve, it is Adam to win immediately if the current position in the word is the last one; thus state q ma be thought of as non-accepting in that case.

Remark 2.1. Alternating register automata, as defined above, are expressively equivalent to the automata known in literature, see e.g. (Demri and Lazić, 2009). On the other hand, our definition of alternating timed automata is expressively equivalent to the definitions of (Lasota and Walukiewicz, 2005, 2008) and (Ouaknine and Worrell, 2005). The equivalence boils down to subclasses of deterministic and nondeterministic automata.

2.4. *Isomorphisms*

By a *time isomorphism* we mean any order-preserving bijection f over the interval $[0, 1)$ (this implies $f(0) = 0$ in particular). The intuition is that an isomorphism will not be applied to a time stamp t, but to its fractional part only (that we write $\widehat{t}$), keeping the integer part $\lfloor t \rfloor$ unchanged.

Given a time isomorphism f, we apply it to a timed word $w = (a_1, t_1) \cdots (a_n, t_n)$ as follows:

$$f(w) = (a_1, \lfloor t_1 \rfloor + f(\widehat{t_1}))(a_2, \lfloor t_2 \rfloor + f(\widehat{t_2})) \cdots (a_n, \lfloor t_n \rfloor + f(\widehat{t_n})).$$

Proposition 2.2. Languages recognized by alternating timed automata are closed under time isomorphism: for any timed automaton $\mathcal{A}$ and a time isomorphism f, $\mathcal{A}$ accepts a timed word w iff $\mathcal{A}$ accepts $f(w)$.

Proof. For a given a timed word w accepted by $\mathcal{A}$, the automaton can only make tests for labels, or tests of whether for two positions i, j of w, $t_j - t_i \sqsubseteq k$ for some $k \in \mathbb{N}$, for $\sqsubseteq \in \{<, \leq, >, \geq, =\}$. Note that $t_j - t_i = (\lfloor t_j \rfloor + \widehat{t_j}) - (\lfloor t_i \rfloor + \widehat{t_i}) = (\lfloor t_j \rfloor - \lfloor t_i \rfloor) + (\widehat{t_j} - \widehat{t_i})$, where $\widehat{t_j} - \widehat{t_i} \in [0, 1)$. Since f is bijective and order preserving, $f(\widehat{t_j}) - f(\widehat{t_i}) \in [0, 1)$, and $\widehat{t_j} - \widehat{t_i} = 0$ if, and only if, $f(\widehat{t_j}) - f(\widehat{t_i}) = 0$. Hence, $t_j - t_i \sqsubseteq k$ if, and only if, $(\lfloor t_j \rfloor + f(\widehat{t_j})) - (\lfloor t_i \rfloor + f(\widehat{t_i})) \sqsubseteq k$. This, added to the fact that $f(w)$ preserves the labels of w, implies that if w is accepted by $\mathcal{A}$, $f(w)$ is also accepted by $\mathcal{A}$. The reciprocal also holds, since $w = f^{-1}(f(w))$. □

We say that two data words $(a_1, d_1)(a_2, d_2) \ldots (a_n, d_n)$ and $(a_1, e_1)(a_2, e_2) \ldots (a_n, e_n)$

with the same string projection $a_1 a_2 \dots a_n$ are *data isomorphic* if for all $i, j \in \{1 \dots n\}$, $d_i \preceq d_j$ iff $e_i \preceq e_j$.

Proposition 2.3. Languages recognized by alternating register automata are closed under data isomorphism: for any register automaton $\mathcal{A}$ and two data isomorphic words w and v, $\mathcal{A}$ accepts w iff $\mathcal{A}$ accepts v.

3. Braids

An idea which is crucial to obtain reductions in both directions is an instrumentation of timed and data words, to be defined in this section, that enforces a kind of 'braid' structure in a word.

3.1. *Data braids*

The *data projection* of $w = (a_1, d_1) \dots (a_n, d_n) \in (\mathbb{A} \times \mathbb{D})^*$ is $d_1 \dots d_n \in \mathbb{D}^*$. We define the *ordered partition* of a data word w as a factorization

$$w_1 \cdot \ldots \cdot w_k = w \tag{3}$$

into data words $w_1, \dots, w_k$ such that each w_i is a maximal subword strictly ordered with respect to $\prec$. In other words: all the data values of any w_i are strictly increasing, and for all $i < k$, the first data value of w_{i+1} is less or equal to the last one of w_i. It follows that for every data word there is a unique ordered partition.

A data word w is a *data braid* iff

— The minimum data value of w appears at the first position.
— Its ordered partition is such that the data projection of each factor w_i is a substring of the data projection of w_{i+1}. In this context, we say that v is a *substring* of v' iff v is the result of removing some (possibly none) positions from v'.
— We can partition the alphabet $\mathbb{A} = \mathbb{A}_1 \cup \mathbb{A}_2$ so that a position i of w is labeled with a symbol of $\mathbb{A}_2$ iff $d_i = d_1$. We call a *marked position* to any $\mathbb{A}_2$-labeled position of the word. Note that the marked positions are those starting some factor of the ordered partition of w.

Example 3.1. The word w below is not an ordered data braid since its ordered partition does not satisfy the substring requirement. Neither is v, since the minimum element does not appear at the first position. In this example as well as in the following ones we use natural number as exemplary data value.

$$\begin{aligned} w &= (c,1) \cdot (d,1)(a,4)(b,8) \cdot (c,1)(b,2)(a,4)(a,8)(b,9) \cdot (c,1), \\ v &= (c,3) \cdot (d,2)(a,3)(b,8) \cdot (c,2)(b,3)(a,5)(a,8). \end{aligned}$$

In the case of w, the substring requirement is fulfilled if, e.g., the last element $(c,1)$ is removed, or when w is extended with $(b,2)(a,4)(b,5)(a,8)(b,9)$; in both cases $\mathbb{A}_1 = \{a, b\}$ and $\mathbb{A}_2 = \{c, d\}$.

3.2. *Timed braids*

Intuitively, the braid condition for timed words is analogous to that of ordered data braids if one considers the fractional part of a time stamp t_i as datum. A timed word

$$w = (a_1, t_1)(a_2, t_2) \dots (a_n, t_n)$$

is a *timed braid* iff

— The first time stamp equals zero, $t_1 = 0$.
— For all $i < n$, if a time-stamp with integer part greater than $\lfloor t_i \rfloor$ appears in w, then the time-stamp $t_i + 1$ appears in w.
— The alphabet can be partitioned into $\mathbb{A} = \mathbb{A}_1 \cup \mathbb{A}_2$ so that the marked positions (i.e., those labeled by $\mathbb{A}_2$) are precisely those carrying an *integer* time stamp.

Braids will play a central role in the following section. In fact both data braids and timed braids represent essentially the same concept, disregarding some minor details, as illustrated next.

Example 3.2. We show a data braid w and a 'corresponding' timed braid v. $\mathbb{A}_1 = \{a, b\}$ and $\mathbb{A}_2 = \{\bar{a}, \bar{b}\}$.

$$\begin{array}{rcccccc} w & = & (\bar{b}, 2)(a, 4) & \cdot & (\bar{a}, 2)(b, 4)(b, 8) & \cdot & (\bar{b}, 2)(b, 3)(a, 4)(a, 8)(b, 9) \\ v & = & (\bar{b}, 0.0)(a, 0.5) & \cdot & (\bar{a}, 1.0)(b, 1.5)(b, 1.6) & \cdot & (\bar{b}, 2.0)(b, 2.3)(a, 2.5)(a, 2.6)(b, 2.9). \end{array}$$

The particular data values and time stamps are exemplary ones. A canonical way of obtaining a timed braid from a data braid (and vice versa), to be explained below, will be ambiguous up to time (data) isomorphism.

3.3. *Transformations*

We introduce two simple encodings: one maps a timed word into a timed braid, and the other maps a data word into a data braid.

$$\begin{array}{ccc} \text{timed words} & \longrightarrow & \text{timed braids} \\ & & \updownarrow \\ \text{data words} & \longrightarrow & \text{data braids} \end{array}$$

A timed word w over an alphabet $\mathbb{A}$ induces a timed braid $\mathrm{tb}(w)$ over the extended alphabet $\mathbb{A} \cup \{\checkmark\} \cup \bar{\mathbb{A}} \cup \{\bar{\checkmark}\}$, where $\bar{\mathbb{A}} = \{\bar{a} \mid a \in \mathbb{A}\}$, as follows. First, if $t_1 \neq 0$, add the pair $(\checkmark, 0)$ at the very first position. Then add pairs $(\checkmark, t)$ at all time points t that are missing according to the definition of timed braid. Finally change every symbol a at each position carrying an integer time stamp by its 'marked' counterpart $\bar{a} \in \bar{\mathbb{A}} \cup \{\bar{\checkmark}\}$.

A data word w over $\mathbb{A}$ may be canonically extended to a data braid $\mathrm{db}(w)$ over the alphabet $\mathbb{A} \cup \{\checkmark\} \cup \bar{\mathbb{A}} \cup \{\bar{\checkmark}\}$ as follows. Consider the ordered partition $w = w_1 \cdot \ldots \cdot w_n$ and let $d_{\min}$ be the smallest datum appearing in w. Firstly, for every factor w_i, add the pair $(\checkmark, d_{\min})$ at the very first position of w_i, unless w_i already contains the datum $d_{\min}$. Secondly, for each datum d appearing in any w_i, add $(\checkmark, d)$ to each of the following

factors $w_{i+1} \dots w_n$ that do not contain d. This insertion is done preserving the order of the factor. Finally, change every symbol a at the first position of a factor by its 'marked' counterpart $\bar{a} \in \bar{\mathbb{A}} \cup \{\bar{\checkmark}\}$. Note that as a result we obtain a data braid.

Example 3.3. As an illustration, consider the effect of the above transformations on an exemplary data word w and a timed word v.

$$\begin{aligned} w &= (a,4) \quad \cdot \quad (b,1)(a,4)(b,8) \quad \cdot \quad (a,1)(a,5)(a,8) \\ \mathrm{db}(w) &= (\bar{\checkmark},1)(a,4) \quad \cdot \quad (\bar{b},1)(a,4)(b,8) \quad \cdot \quad (\bar{a},1)(\checkmark,4)(a,5)(a,8) \\ v &= (a,0.0)(a,0.7) \quad \cdot \quad (b,1.5) \quad \cdot \quad (b,2.0) \\ \mathrm{tb}(v) &= (\bar{a},0.0)(a,0.7) \quad \cdot \quad (\bar{\checkmark},1.0)(b,1.5)(\checkmark,1.7) \quad \cdot \quad (\bar{b},2.0)(\checkmark,2.5)(\checkmark,2.7) \end{aligned}$$

We have thus explained the horizontal arrows of the diagram, and now we move to the vertical ones. Both vertical mappings preserve the length of the word.

A timed braid $(a_1,t_1)\dots(a_n,t_n)$ gives naturally rise to a data braid by replacing each time stamp t_i by its fractional part $\widehat{t_i}$, and then mapping the set $\{\widehat{t_1},\dots,\widehat{t_n}\}$ into the data domain $\mathbb{D}$ through an order-preserving injection. We only want to consider order-preserving injections, thus this always yields a data braid. Note that the choice of a particular order-preserving injection is irrelevant, as one always obtains the same data word up to data isomorphism (cf. Proposition 2.3). We hope this ambiguity will not be confusing.

A data braid $w = (a_1,d_1)\cdots(a_n,d_n)$ may be turned into a timed braid through any order-preserving injection $f : \{d_1,\dots,d_n\} \to [0,1)$ such that $f(d_1) = 0$. Each element (a_i,d_i) is mapped into a similar element $(a_i, k + f(d_i))$, where k is the number of factors (in the ordered partition of w) that end strictly before position i. Consecutive factors will get consecutive natural numbers as the integer part of time stamps. As before, we consider the choice of a particular injection f irrelevant (cf. Proposition 2.2).

Notice that going from a timed braid to a data braid and back returns to the original word up to time isomorphism; likewise, combining the transformations in the reverse order we get back to the same word, up to data isomorphism.

Slightly overloading the notation, we write $\mathrm{db}(w)$ to denote the data braid obtained from a *timed* word w by the appropriate composition of transformations just described. Similarly, we write $\mathrm{tb}(w)$ to denote the timed braid obtained from a *data* word w.

4. From timed automata to register automata

We are going to show that, up to a suitable encoding, languages recognized by timed automata are recognized by register automata as well. The transformation keeps the number of registers equal to the number of clocks, and preserves the mode of computation (nondeterministic, co-nondeterministic, alternating).

Theorem 4.1. Given an alternating timed automaton $\mathcal{A}$ one can compute in exponential time an order-blind register automaton $\mathcal{B}$ such that for any timed word w, $\mathcal{A}$ accepts w if an only if $\mathcal{B}$ accepts $\mathrm{db}(w)$. The number of registers of $\mathcal{B}$ equals the number of

clocks of $\mathcal{A}$. Moreover, $\mathcal{B}$ is deterministic (resp. nondeterministic, co-nondeterministic, alternating) if $\mathcal{A}$ is so.

Proof. We describe the construction of a register automaton $\mathcal{B}$ that faithfully simulates a given timed automaton $\mathcal{A}$. The idea is that the behavior of each clock can be simulated by a register. When the clock is reset on one automaton, the other loads the current data value d into the register. Then, by the data braid structure, the register automaton knows exactly how many units of time have elapsed for the clock by simply counting the number of times that d has appeared.

Consider the maximum constant $k_{\max}$ that appears in the transitions of $\mathcal{A}$. Let Q and $\mathcal{C}$ denote the states and clocks of $\mathcal{A}$, respectively. The states of $\mathcal{B}$ will be $Q' \times \{0, \{0,1\}, 1, \{1,2\}, \ldots, k_{\max}, \{k_{\max}, \infty\}\}^{\mathcal{C}}$. The first component Q' will be a superset of Q containing a number of additional auxiliary states. The second component of state will be called *clock component*. Intuitively, for each clock c the automaton $\mathcal{B}$ stores the information about the single-clock region of the current value of c, up to $k_{\max}$. One may distinguish *point* regions and *interval* regions of $v(c)$. The initial state is (q_0, v_0) where v_0 assigns 0 to each $c \in \mathcal{C}$. The owner of a state (q, v) in $\mathcal{B}$ will be the same as the owner of q in $\mathcal{A}$, if $q \in Q$, and irrelevant otherwise.

There will be as many registers in $\mathcal{B}$ as clocks in $\mathcal{A}$, $\mathcal{R} = \{r_c | c \in \mathcal{C}\}$, and each register r_c will be used to update the information about the region of c. The integer part of the value of a clock c in $\mathcal{A}$ will correspond to the number of times (up to $k_{\max}$) the value stored in register r_c in $\mathcal{B}$ appeared in the word since it was loaded. Whenever the clock c is *reset* in $\mathcal{A}$, the corresponding action of $\mathcal{B}$ is to *load* the current value to r_c and to change state from (q, v) to $(q, v[c := 0])$. (Recall that it is assumed that as the very first step the automaton $\mathcal{B}$ loads the first data value in the word into all registers.) The automaton $\mathcal{B}$ will also be capable to detect that the integer part of a clock c increases, using the equality test '$= r_c$' as outlined below.

The automaton $\mathcal{B}$, to be described now in more detail, will not distinguish marked symbols from unmarked ones. We consider an arbitrary transition $q \xrightarrow{o} p$ of $\mathcal{A}$, separately for every kind of operation o, and outline the transitions, it gives rise to, in automaton $\mathcal{B}$ from a state (q, v).

Operation o is 'test if the current label is a'. The automaton $\mathcal{B}$ tests that the current label is a or $\bar{a}$, and changes state to (p, v). This does not depend on v. This applies to $a \neq \checkmark$. The labels $\checkmark$ and $\bar{\checkmark}$ are essentially ignored by $\mathcal{B}$, i.e., if the current label is any of the two, the automaton goes to next position without changing state.

Operation o is 'go to next position'. When going to the next position, independently of its (not yet read) label, the automaton $\mathcal{B}$ will update the clock component of its state (q, v). First, for all clocks c with point region k the value of v is changed to $(k, k+1)$ or $(k_{\max}, \infty)$:

$$(q, v) \xrightarrow{o} (q', v^{\rightarrow}),$$

for an auxiliary state $q' \in Q' \setminus Q$ and for $v^{\rightarrow}$ defined by

$$v^{\rightarrow}(c) = \begin{cases} (k, k+1) & \text{if } v(c) = k < k_{\max} \\ (k_{\max}, \infty) & \text{if } v(c) = k_{\max} \\ v(c) & \text{otherwise.} \end{cases}$$

After this, all regions $v(c)$ are intervals. Then, starting from state $(q', v^{\rightarrow})$ the automaton $\mathcal{B}$ performs a sequence of equality checks '$= c$', one for every clock c, possibly updating the value of $v^{\rightarrow}$ on c, according to the following transitions:

$$(q', u) \xrightarrow{=c} (q'', u^c) \qquad\qquad (q', u) \xrightarrow{\neq c} (q'', u),$$

where u^c differs from u only on c, according to:

$$u^c(c) = \begin{cases} k & \text{if } u(c) = (k-1, k) \\ u(c) & \text{otherwise.} \end{cases}$$

This involves a sequence of auxiliary states $q', q'' \in Q' \setminus Q$, etc, whose owner is the same as the owner of q, except for the very last one that we assume to be p.

So far we took into account the cases when the label is not $\checkmark$ or $\bar{\checkmark}$. The idea is that the automaton ignores the $\checkmark$ and $\bar{\checkmark}$ labels, but its clock component must be updated as for other labels. In this case we proceed just as before, but in the last transition, instead of moving to p, it moves to a new state p', and from there back to q' by moving to the next position as described above. In this way, all the $\checkmark$ and $\bar{\checkmark}$ will be skipped, but the clock component of the state will be updated.

Operation o is 'test if $c \sqsubseteq k$'. According the value $v(c)$ of the clock component v, either the clock constraint $c \sqsubseteq k$ is satisfied and then we add a transition

$$(q, v) \longrightarrow (p, v)$$

to $\mathcal{B}$; or the clock constraint is not satisfied, and then there is no transition from (q, v) that corresponds to $q \xrightarrow{o} p$.

Operation o is 'reset clock c'. The corresponding action of $\mathcal{B}$ is to *load* the current value to r_c and to change state from (q, v) to $(q, v[c := 0])$.

Note that each transition $q \xrightarrow{o} p$ of $\mathcal{A}$ may give raise to a number of new states $q, q', q'', \cdots \in Q' \setminus Q$ used in $\mathcal{B}$. However, the total number of states added is exponential in the alphabet $\mathbb{A}$ and the clocks $\mathcal{C}$. Thus, the construction is exponential. No nondeterministic or alternating transitions are explicitly added in this construction, and hence $\mathcal{B}$ is deterministic (resp. nondeterministic, co-nondeterministic, alternating) whenever $\mathcal{A}$ is so.

The automaton $\mathcal{B}$ is order-blind as required. □

Remark 4.2. Observe that the algorithm is exponential only in the number of clocks of the timed automaton. Thus, if the number of clocks is constant, the translation is done in polynomial time.

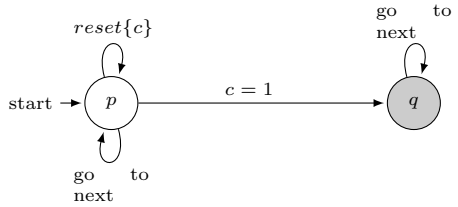


Fig. 1. An automaton checking that there are two time-stamp whose difference is 1. Gray states are owned by Adam and the others by Eve.

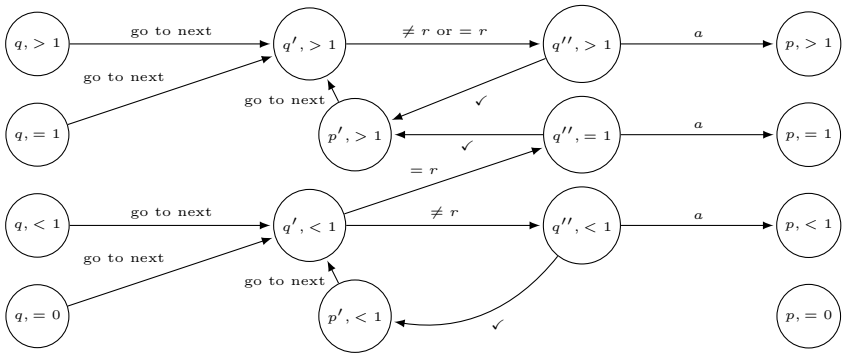


Fig. 2. The 'go to next(q, p)' black box. These transitions correspond to the transformation of $q \xrightarrow{\text{go to next}} p$, assuming that regions are $0, (0,1), 1, (1,\infty)$. This transformation will be used as black box.

Example 4.3. To illustrate the construction, consider the nondeterministic one clock timed automaton that checks that there are two time stamps whose difference is 1, as depicted in Figure 1.

The construction described in the proof of Theorem 4.1, applied to the automaton shown in Figure 1, yields the order-blind register automaton depicted in Figure 3. For convenience of presentation, we abstract the transitions corresponding to a transition $q \xrightarrow{\text{go to next}} p$ as a black box parametrized by q and p. This construction is depicted in Figure 2.

For the successive results, we make use of the following lemma.

Lemma 4.4. The complement of the language of all data braids is recognized by a nondeterministic one register automaton.

Proof. A data word $w = (a_1, d_1) \cdots (a_n, d_n)$ fails to be a data braid iff either

(1) some datum strictly smaller than d_1 appears in w,
(2) there is some marked (i.e., carrying an alphabet letter from $\bar{\mathbb{A}} \cup \bar{\checkmark}$) position i such that $d_i \succ d_1$,
(3) there is some unmarked position i such that $d_i = d_i$,

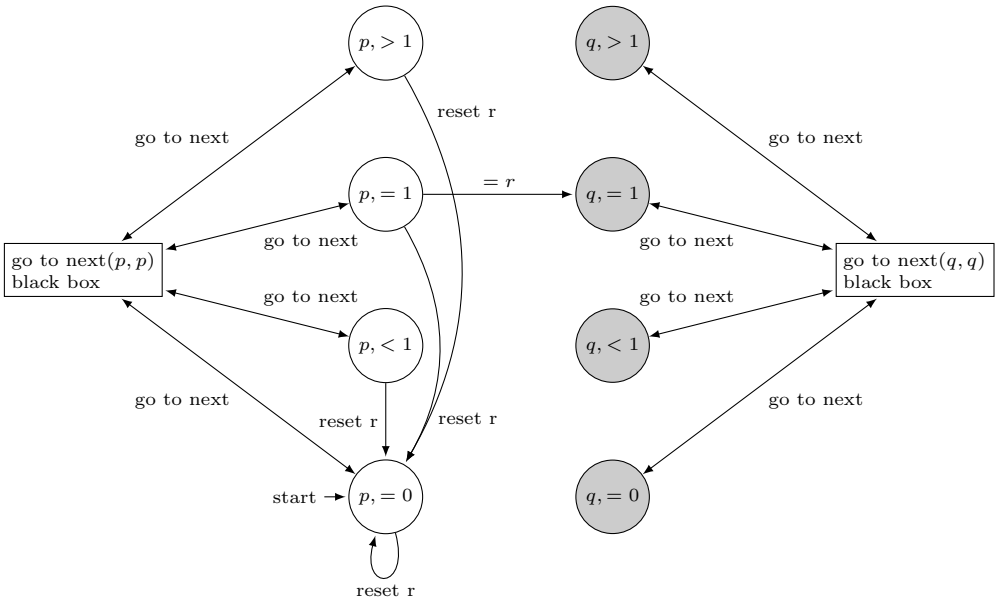


Fig. 3. The automaton resulting from the proof of Theorem 4.1.

(4) for some position i, there are two marked positions $j < k$, both greater than i, such that d_i does not appear among $\{d_j \dots d_{k-1}\}$, or
(5) for some position i there is a marked position $j > i$ such that d_i does not reappear at any position greater or equal j.

A nondeterministic automaton can easily guess which of these conditions holds and verify it using one register. □

As a consequence of Lemma 4.4, we get:

Corollary 4.5. The language of data braids is recognized by a co-nondeterministic one register automaton.

Proof. It suffices to swap the ownership of states between Adam and Eve. □

We want to use Theorem 4.1 together with Lemma 4.4 to show Theorem 4.8 below that reduces decision problems for timed automata to the analogous decision problems for register automata. However, there is a subtle point here: by Lemma 4.4 register automata can recognize the complement of data braids, while we would need register automata to recognize the complement of the *image* of db(_). The latter is a different language, since db(_) is not surjective (for example, consider appending $(\checkmark, d)$ for a sufficiently big d at the end of a data braid), and unfortunately, cannot be recognized by a nondeterministic 1-register automaton. In the proof below we deal with this problem by observing that db(_) is *essentially* surjective onto data braids.

Definition 4.6. A position i in a data braid is considered *useless* iff it is labeled by $(\checkmark, d)$, for some datum d, and moreover

(a) all appearances of the datum d before i are labeled with $\checkmark$; or
(b) all the positions in its factor and in all the following factors are labeled exclusively with $\checkmark$ or $\bar{\checkmark}$.

A data braid with no useless positions we call *non-redundant*.

Note that from any data braid w one obtains a non-redundant one by simultaneously removing all useless positions from w. Denote this non-redundant braid by $\widetilde{w}$.

Lemma 4.7. The mapping $\mathrm{db}(_)$ is essentially surjective onto data braids in the following sense:

(i) $\mathrm{db}(_)$ is a bijection between the isomorphism classes of timed words and non-redundant data braids,
(ii) any automaton $\mathcal{B}$ constructed in Theorem 4.1 can not tell a difference between w and $\widetilde{w}$.

Proof. (i) We claim that every non-redundant data braid w equals to $\mathrm{db}(v)$, up to isomorphism, for some timed word v. Indeed, consider any order preserving injection f from data values appearing in w to $[0, 1)$ that maps the least value to 0. Let v be the result of the following steps: (1) replace every (a_i, d_i) of w with $(a_i, k + f(d_i))$, where k is the number of factors in w that end before position i; (2) remove all $\checkmark/\bar{\checkmark}$ positions; and (3) project the alphabet into $\mathbb{A}$.

(ii) We argue that $\mathcal{B}$ either accepts both a data braid w and $\widetilde{w}$, or none of them. This is true by construction, since when the input letter is $\checkmark$, the register automaton $\mathcal{B}$ only updates the information about the integer part of those clocks c for which the equality test $=_{r_c}$ holds. When reading a useless position, if it is useless because of (a), then the equality holds for no clock; whereas in case (b) the update will be inessential for the acceptance. □

A wide range of decision problems for timed automata reduce to the analogous problems for register automata. This is due to the very tight correspondence between timed words and data braids as stated in Lemma 4.7. We mean here those decision problems whose input is a number of timed automata $\mathcal{A}_1 \dots \mathcal{A}_k$ and the question is about the languages $\mathcal{L}(\mathcal{A}_1) \dots \mathcal{L}(\mathcal{A}_k)$.

We believe that it is not instructive to formally state the reduction in full generality. Instead, we prefer to consider one chosen problem, to illustrate the method of reasoning. As an illustrative example we consider the inclusion problem that for given $\mathcal{A}$ and $\mathcal{B}$ asks if $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$. This particular problem, being undecidable in general, has interesting decidable special cases, for example when when $\mathcal{B}$ is a 1-clock alternating automaton and $\mathcal{A}$ is assumed to be nondeterministic (but allowed to have arbitrarily many clocks). Thus it will be extremely important that the reduction preserves the number of clocks/registers and the mode of computation, as it was the case in Theorem 4.1.

The discussion above motivates the following restricted version of the inclusion problem. Let $\mathfrak{A}$ and $\mathfrak{B}$ be two classes of timed automata. Each of these classes is determined

by a restriction on the mode of computation and/or on the number of clocks. Note that the same kind of restrictions makes equally sense for register automata, but applies to the number of registers instead of clocks. We write $\mathfrak{A}^r$ and $\mathfrak{B}^r$ to the corresponding classes of register automata. By $\mathfrak{A} \subseteq \mathfrak{B}$ we mean a restricted inclusion problem, that asks if $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ for given $\mathcal{A} \in \mathfrak{A}$ and $\mathcal{B} \in \mathfrak{B}$.

We say that a class of automata is *effectively closed under union/intersection* if for two given automata from that class, an automaton may be computed that belongs to the class and recognizes the union/intersection of the languages of the two given automata. For instance, k-clock/register nondeterministic automata are effectively closed under union and k-clock/register co-nondeterministic automata are effectively closed under intersection.

Theorem 4.8. The restricted inclusion problem $\mathfrak{A} \subseteq \mathfrak{B}$ for timed automata reduces, in exponential time, to the analogous problem $\mathfrak{A}^r \subseteq \mathfrak{B}^r$ for register automata, whenever any of the following conditions hold:

— class $\mathfrak{B}$ contains nondeterministic 1-clock automata and is effectively closed under union, or
— class $\mathfrak{A}$ contains co-nondeterministic 1-clock automata and is effectively closed under intersection.

Proof. For two given timed automata $\mathcal{A} \in \mathfrak{A}$ and $\mathcal{B} \in \mathfrak{B}$, the problem asks if there is a timed word w such that

$$w \in \mathcal{L}(\mathcal{A}) \quad \text{and} \quad w \notin \mathcal{L}(\mathcal{B}).$$

By Theorem 4.1 we get two register automata $\mathcal{A}'$ and $\mathcal{B}'$ such that the question is equivalent to:

$$\mathrm{db}(w) \in \mathcal{L}(\mathcal{A}') \quad \text{and} \quad \mathrm{db}(w) \notin \mathcal{L}(\mathcal{B}').$$

By Lemma 4.7 it is equivalent to ask if there is a data braid w such that

$$w \in \mathcal{L}(\mathcal{A}') \quad \text{and} \quad w \notin \mathcal{L}(\mathcal{B}').$$

If the assumption concerning $\mathfrak{B}$ holds, we observe that it is equivalent to ask if there is a data word w such that

$$w \in \mathcal{L}(\mathcal{A}') \quad \text{and} \quad w \notin \mathcal{L}(\mathcal{B}') \quad \text{and} \quad w \notin \mathcal{L}(\mathcal{A}_{\neg\mathrm{db}}),$$

for a nondeterministic 1-register automaton $\mathcal{A}_{\neg\mathrm{db}}$ given by Lemma 4.4, which rewrites to

$$w \in \mathcal{L}(\mathcal{A}') \quad \text{and} \quad w \notin (\mathcal{L}(\mathcal{B}') \cup \mathcal{L}(\mathcal{A}_{\neg\mathrm{db}})). \tag{4}$$

Otherwise, if the assumption concerning $\mathfrak{A}$ holds, the question is equivalent to asking if there is a data word such that

$$w \in \mathcal{L}(\mathcal{A}') \quad \text{and} \quad w \in \mathcal{L}(\mathcal{A}_{\mathrm{db}}) \quad \text{and} \quad w \notin \mathcal{L}(\mathcal{B}'),$$

for a co-nondeterministic 1-register automaton $\mathcal{A}_{\mathrm{db}}$ existing by Corollary 4.5, which rewrites to

$$w \in (\mathcal{L}(\mathcal{A}') \cap \mathcal{L}(\mathcal{A}_{\mathrm{db}})) \quad \text{and} \quad w \notin \mathcal{L}(\mathcal{B}'). \tag{5}$$

In both cases (4) and (5), we arrive at an instance of the corresponding inclusion problem $\mathfrak{A} \subseteq \mathfrak{B}$ for register automata. As these instances are obtained effectively, the reduction if thus proved. □

We claim that many other decision problems, e.g., non-emptiness, universality, or equality, concerning languages, their intersections, unions, etc., may be treated in precisely the same way.

5. From register automata to timed automata

In this section we complete the relation between the models of automata. We show that, up to a suitable encoding, languages of register automata may be recognized by timed automata. Again, this transformation keeps the number of registers equal to the number of clocks, and preserves the mode of computation (nondeterministic, co-nondeterministic, alternating). Thus we obtain a tight relationship between the two classes of automata.

Theorem 5.1. Given an alternating register automaton $\mathcal{A}$ one can compute in exponential time a timed automaton $\mathcal{B}$ such that for any data word w, $\mathcal{A}$ accepts w if and only if $\mathcal{B}$ accepts $\mathrm{tb}(w)$. The number of clocks of $\mathcal{B}$ equals the number of registers of $\mathcal{A}$. Moreover, $\mathcal{B}$ is deterministic (resp. nondeterministic, co-nondeterministic, alternating) if $\mathcal{A}$ is so.

Proof. We describe the construction of a timed automaton $\mathcal{B}$ that faithfully simulates the behavior of a given register automaton $\mathcal{A}$. Let $\mathcal{R}$ be the set of registers of $\mathcal{A}$. The number of clocks in $\mathcal{B}$ is the same as the number of registers in $\mathcal{A}$, $\mathcal{C} = \{c_r | r \in \mathcal{R}\}$. A clock c_r is reset whenever $\mathcal{A}$ loads the current data value into register r. Moreover, each clock is also reset whenever the constraint $c_r = 1$ is met. Thus, when $\mathcal{B}$ runs over a time braid, no clock will ever have value greater than 1.

The state space of $\mathcal{B}$ is built on top of the states Q of $\mathcal{A}$, extended with a number of additional auxiliary states. Additionally, for each clock c_r the automaton $\mathcal{B}$ stores in its state one bit of information describing whether the last marked position was seen before or after the last reset of c_r. This will allow $\mathcal{B}$ to simulate tests comparing the current data value with data values stored in registers.

Formally, states of $\mathcal{B}$ are pairs $(q, X) \in Q' \times \mathcal{P}(\mathcal{R})$, for $Q \subseteq Q'$. Initially the initial state of $\mathcal{A}$ is chosen and the set X, which we call the *register component* of a state, is chosen as $X = \emptyset$. At each marked symbol $\bar{a}$ or $\bar{\surd}$, the automaton $\mathcal{B}$ sets $X := \emptyset$. Moreover, at each reset of r (at marked or unmarked positions), r is added to X. As a consequence of this behavior, it invariantly holds: $r \in X$ if and only if the position of the last reset of c_r is greater or equal to the last marked position. Hence, the test $\preceq r$ (current data smaller or equal to register r) is satisfied at a state (q, X) if and only if $r \notin X$. The table below summarizes all the data tests and the corresponding constraints on clock values and on the register component of state:

test in $\mathcal{A}$	meaning	constraint in $\mathcal{B}$
$\succ r$	current datum greater than r	$r \in X$ and $c_r > 0$
$\succeq r$	current datum greater or equal to r	$r \in X$
$\prec r$	current datum smaller than r	$r \notin X$
$\preceq r$	current datum smaller or equal to r	$r \notin X$ or $c_r = 0$

We consider an arbitrary transition $q \xrightarrow{o} p$ of $\mathcal{A}$, separately for every kind of operation o, and outline transitions of $\mathcal{B}$ it gives rise to.

Operation o is 'test if the current label is a'. In state (q, X) the automaton $\mathcal{B}$ tests if the current label is a or $\bar{a}$, and changes state to (p, X). This does not depend on X and applies to $a \neq \checkmark$ only. Thus the labels $\checkmark$ and $\bar{\checkmark}$ are essentially ignored by $\mathcal{B}$.

Operation o is 'go to next position'. When going to the next position, the automaton $\mathcal{B}$ will test if it contains a marked label or not, and if so it will empty X. That is, we first add a transition from (q, X) to (q', X) that moves to next position. And we also add transitions from (q', X) to $(q'', \emptyset)$ testing for every possible label $\bar{a} \in \bar{\mathbb{A}}$, and from (q', X) to (q'', X) for every possible label $a \in \mathbb{A}$. Finally, it tests sequentially, for every clock c_r whether $c_r = 1$. If so, it updates X with $X \cup \{r\}$ and resets c_r, otherwise it does not modify X or c_r. All this is implemented by adding, for each clock, two fresh states to $Q' \setminus Q$, whose owner is the same as the owner of q. As a final step, $\mathcal{B}$ moves to state p. For example, assuming we have only one clock r, we would add a transition from (q'', X) to $(q''', X \cup \{r\})$ testing $c_r = 1$, a transition from (q''', X) to (p, X) resetting c_r, and one transition from (q'', X) to (p, X) testing $c_r \neq 1$.

So far we took into account the cases when the label is not $\checkmark$ or $\bar{\checkmark}$. The idea is that the automaton ignores the $\checkmark$ and $\bar{\checkmark}$ labels, but its register component X must be updated accordingly. In this case we proceed just as before, but in the last transition, instead of moving to p, it moves to a new state p', and from there back to q' by moving to the next position. In this way, all the $\checkmark$ and $\bar{\checkmark}$ will be skipped, but the X component of the state will be updated.

Operation o is 'test $\sqsubseteq r$'. If $\sqsubseteq$ is $\succ$, we add a transition from (p, X) to (q, X) testing that '$c_r > 0$', for any X containing r. If $\sqsubseteq$ is $\prec$, $\succeq$ or $\preceq$ we proceed accordingly, using the table seen before.

Operation o is 'load current datum to r'. The corresponding action of $\mathcal{B}$ is to *reset clock* c_r and to change state from (q, X) to $(p, X \cup \{r\})$.

Note that each transition $q \xrightarrow{o} p$ of $\mathcal{A}$ may give raise to a number of new states $q, q', q'', \cdots \in Q' \setminus Q$ used in $\mathcal{B}$. However, the total number of states added is exponential in the alphabet $\mathbb{A}$ and the registers $\mathcal{R}$. Thus, the construction is exponential. No nondeterministic or alternating transitions are explicitly added in this construction, and hence

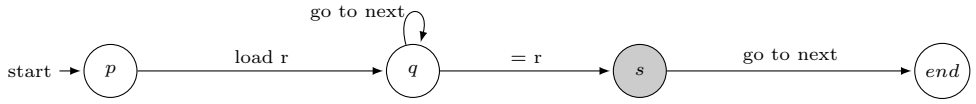


Fig. 4. An automaton checking that the first datum is equal to the last one.

$\mathcal{B}$ is deterministic (resp. nondeterministic, co-nondeterministic, alternating) whenever $\mathcal{A}$ is so. □

Example 5.2. Consider the simple nondeterministic one register automaton that checks if the first datum in a word is equal to the last one depicted in Figure 4.

The construction in the proof of Theorem 5.1 yields the automaton depicted in Figures 5 and 6.

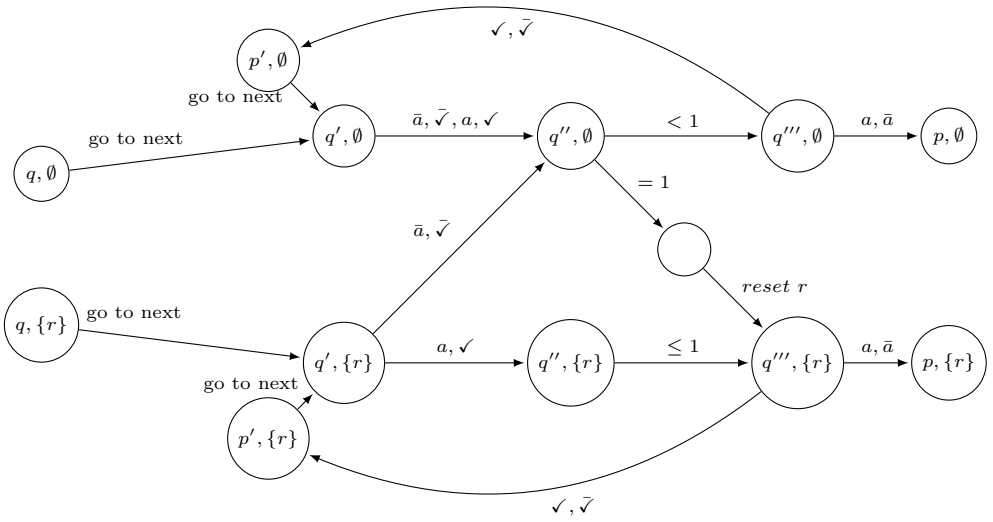


Fig. 5. The 'go to next(q, p)' black box. These transitions correspond to the transformation of $q \xrightarrow{\text{go to next}} p$, assuming that there is only one register. This transformation will be used as black box.

For the next results, we make use of the following lemma.

Lemma 5.3. The complement of the language of all timed braids is recognized by a nondeterministic one clock automaton.

Proof. A timed word fails to be a timed braid iff either

(1) the time-stamp in the first position is not equal 0,
(2) there is some marked position appearing at some non-integer position,
(3) there is some unmarked position appearing at some integer position, or

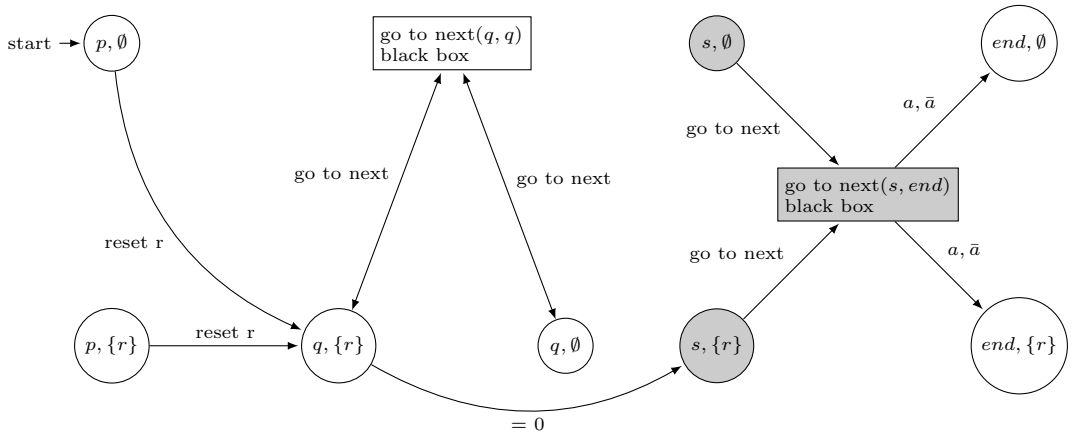


Fig. 6. The result of the construction of Theorem 5.1.

(4) a time-stamp $t + 1.0$, induced by the existence of an element (a, t) for some a, is missing.

It is easy to check that (0), (1) and (2) can be verified by a deterministic one clock automaton. (3) is verified as follows. The automaton guesses a position i, where it resets the clock. Then it checks that after the next marked position, the clock is continuously strictly smaller than 1 until it finally becomes strictly greater than 1, or the word ends. □

As a consequence of Lemma 5.3, we get:

Corollary 5.4. The language of data braids is recognized by a co-nondeterministic one clock automaton.

Proof. It suffices to swap the ownership of states between Adam and Eve. □

Similarly as in Section 4, we want to use Theorem 5.1 together with Lemma 5.3 to show Theorem 5.7 below. We find the same difficulty as before, since timed automata cannot recognize that a timed word is in the *image* of $\mathrm{tb}(_)$. We define a *useless* position and a *non-redundant* timed braid in the same way as for data braids, and we show that $\mathrm{tb}(_)$ is essentially surjective onto timed braids.

Definition 5.5. A position i in a timed braid is considered *useless* iff it is labeled by $(\checkmark, t)$, for some time stamp t, and moreover

(a) every label (a, t') of a position preceding i such that $\widehat{t} = \widehat{t'}$ satisfies $a = \checkmark$; or
(b) all the positions with timestamps in the interval $(t-1, t]$ are labeled exclusively with $\checkmark$ or $\bar{\checkmark}$.

A timed braid with no useless positions is called *non-redundant*.

From any timed braid w one obtains a non-redundant one by applying the following two-step operation:

— simultaneously remove all useless position of type (a) from w; then
— as long as some useless position of type (b) exists, choose one such position, labeled with $(\checkmark, t)$ say, and remove all the positions with time stamp in the interval $(t-1, t]$; then decrease all the following time stamps by 1.

We denote this non-redundant braid obtained from w by $\widetilde{w}$.

Lemma 5.6. The mapping $\mathrm{tb}(_)$ is essentially surjective onto timed braids in the following sense:

(i) $\mathrm{tb}(_)$ is a bijection between the isomorphism classes of data words and non-redundant timed braids,
(ii) any automaton $\mathcal{B}$ constructed in Theorem 5.1 can not tell a difference between w and $\widetilde{w}$.

Proof. (i) We claim that every non-redundant timed braid w equals to $\mathrm{tb}(v)$, up to isomorphism, for some data word v. Indeed, let v be the result of the following steps: (1) replace every (a_i, t_i) of w with $(a_i, \widehat{t_i})$; (2) remove all $\checkmark/\bar{\checkmark}$ positions; and (3) project the alphabet into $\mathbb{A}$.

(ii) We argue that $\mathcal{B}$ either accepts both the timed braid w and $\widetilde{w}$, or none of them. This is true by construction. Note that when reading $(\checkmark, t)$, the register automaton $\mathcal{B}$ does not change the first component of its state, it only updates the information about the register component X. Suppose the position is useless. If it is useless because of (a), then there is no clock c_r where $c_r = 1$, because no clock was ever reset in any previous position with the same fractional part $\widehat{t}$. Hence, the register component X of the state does not change. In the case of (b), the update will be inessential for the acceptance. □

The following theorem is proved analogously to Theorem 4.8, using Theorem 5.1, Lemma 5.3, and Lemma 5.6:

Theorem 5.7. The restricted inclusion problem $\mathfrak{A} \subseteq \mathfrak{B}$ for register automata reduces to the analogous problem for timed automata, whenever any of the following conditions hold:

— class $\mathfrak{B}$ contains nondeterministic 1-register automata and is effectively closed under union, or
— class $\mathfrak{A}$ contains co-nondeterministic 1-register automata and is effectively closed under intersection.

Proof. As expected, the proof is completely analogous to the proof of Theorem 4.8, we include it for the sake of completeness. For two given register automata $\mathcal{A} \in \mathfrak{A}$ and $\mathcal{B} \in \mathfrak{B}$, the problem asks if there is a data word w such that

$$w \in \mathcal{L}(\mathcal{A}) \quad \text{and} \quad w \notin \mathcal{L}(\mathcal{B}).$$

By Theorem 5.1 we get two register automata $\mathcal{A}'$ and $\mathcal{B}'$ such that the question is equivalent to:

$$\mathrm{tb}(w) \in \mathcal{L}(\mathcal{A}') \quad \text{and} \quad \mathrm{tb}(w) \notin \mathcal{L}(\mathcal{B}').$$

By Lemma 5.6 it is equivalent to ask if there is a timed braid w such that

$$w \in \mathcal{L}(\mathcal{A}') \text{ and } w \notin \mathcal{L}(\mathcal{B}').$$

If the assumption concerning $\mathfrak{B}$ holds, we observe that it is equivalent to ask if there is a data word w such that

$$w \in \mathcal{L}(\mathcal{A}') \text{ and } w \notin \mathcal{L}(\mathcal{B}') \text{ and } w \notin \mathcal{L}(\mathcal{A}_{\neg\mathrm{tb}}),$$

for a nondeterministic 1-clock automaton $\mathcal{A}_{\neg\mathrm{tb}}$ given by Lemma 5.3, which rewrites to

$$w \in \mathcal{L}(\mathcal{A}') \text{ and } w \notin (\mathcal{L}(\mathcal{B}') \cup \mathcal{L}(\mathcal{A}_{\neg\mathrm{tb}})). \tag{6}$$

Otherwise, if the assumption concerning $\mathfrak{A}$ holds, the question is equivalent to asking if there is a data word such that

$$w \in \mathcal{L}(\mathcal{A}') \text{ and } w \in \mathcal{L}(\mathcal{A}_{\mathrm{tb}}) \text{ and } w \notin \mathcal{L}(\mathcal{B}'),$$

for a co-nondeterministic 1-clock automaton $\mathcal{A}_{\mathrm{tb}}$ existing by Corollary 5.4, which rewrites to

$$w \in (\mathcal{L}(\mathcal{A}') \cap \mathcal{L}(\mathcal{A}_{\mathrm{tb}})) \text{ and } w \notin \mathcal{L}(\mathcal{B}'). \tag{7}$$

In both cases (6) and (7), we arrive at an instance of the corresponding inclusion problem $\mathfrak{A} \subseteq \mathfrak{B}$ for timed automata. As these instances are obtained effectively, the reduction if thus proved. □

In the same way one may treat other decision problems, such as non-emptiness, universality or equality.

6. Infinite words

So far we have only considered automata over finite words. In this section we explain how our results may be lifted to automata over ω-words with weak or strong parity acceptance condition. To avoid a lengthy presentation, we restrict ourselves here to a brief discussion of modifications necessary to accommodate ω-words.

For the purpose of this section it is necessary to assume that the data order $\preceq$ is *dense*.

6.1. *Automata over infinite words*

A data ω-word is an element of $(\mathbb{A} \times \mathbb{D})^\omega$, cf. (1). Similarly, a timed ω-word is an element of $(\mathbb{A} \times \mathbb{R}_+)^\omega$ whose time-stamps are strictly monotonic, see (2).

Timed and register automata are as in Sections 2.1 and 2.2, but with one additional ingredient, a rank function $\Omega : Q \to \mathbb{N}$, that assigns a number to each state. The function is used for acceptance of an ω-word in the following way. If a play of the acceptance game is finite, the winner is the same as in the case of finite words. Otherwise, the winner is Eve if the sequence $q_1, q_2, \ldots$ of states appearing in the play satisfies:

— (*weak parity condition*) the smallest number in $\{\Omega(q_i) | i = 1, 2, \ldots\}$ is even.
— (*strong parity condition*) the smallest number appearing infinitely often in the sequence $\Omega(q_1), \Omega(q_2), \ldots$ is even.

We speak of weak/strong (k,l)-parity condition if the ranks are in the range $k \dots l$. Wlog. one may assume $k \in \{0,1\}$. Strong $(0,1)$-parity is traditionally called Büchi condition.

Timed/register automata with weak $(0,0)$-parity condition are called *safety* automata (thus the only way to reject an input word is due to a finite 'bad prefix', which leads to Eve being stuck in its position). Dually, *co-safety* automata are those with weak $(1,1)$-parity condition (thus the only way to accept is when there is some 'good prefix', that leads to Adam being stuck in its position).

6.2. *Infinite braids*

We will distinguish two settings, both for timed and data words. In the first one, we only consider *non-Zeno* words as a legitimate input to an automaton; in the second one, we consider all ω-words. The two settings will be called non-Zeno and unrestricted, respectively.

Recall that a timed ω-word is called non-Zeno when the time-stamps occurring in the word are not bounded; otherwise we call it *Zeno*. We impose an analogous definition for data words: we say that a data ω-word is non-Zeno if the data value decreases infinitely often, i.e., $d_{i+1} \preceq d_i$ for infinitely many positions i. Thus a data ω-word is Zeno if and only if its ordered partition has only finitely many factors (the last one is infinite).

Recall the definitions of data and timed braids in Sections 3.1 and 3.2. We adapt the definition in a straightforward way to ω-words, with one proviso: we intentionally allow Zeno ω-braids in which the last factor does not contain all previously seen data values (or fractional parts of time-stamps), as illustrated in the example below. Intuitively, this naturally corresponds to the fact that in the case of finite words, we could equivalently use prefixes of braids, i.e., we could allow the data projection of last factor to be a prefix of a superstring of the previous one.

Example 6.1. Consider an exemplary Zeno timed ω-word:

$$(\bar{a},0.0)(a,0.7) \cdot (\bar{\checkmark},1.0)(b,1.5)(\checkmark,1.7) \cdot (\bar{b},2.0)(a,2.1)(a,2.15)(a,2.175)(a,2.1875)\dots$$

Even if the fractional parts of time-stamps 1.5 and 1.7 do not reappear in the last infinite factor, it seems natural to include this word into timed ω-braids. Analogous situations may appear in a Zeno data ω-braid, as illustrated by the following one:

$$(\bar{\checkmark},1)(a,4) \quad \cdot \quad (\bar{b},1)(a,4)(b,8) \quad \cdot \quad (\bar{a},1)(\checkmark,4)(a,4.5)(a,4.75)(a,4.875)\dots$$

The transformation of a timed word into a timed braid is exactly as in the case of finite words. Some additional care must be taken in the transformation of a data word w: instead of the smallest data value $d_{\min}$ we use an arbitrary lower bound of all data values appearing in w, and transform w via an isomorphism beforehand if there is no such bound. When going from a timed braid to a data braid, an arbitrary injection of all the fractional parts of time-stamps into $\mathbb{D}$ is used, existing by the density assumption. For the opposite direction, any injection of $\{d \in \mathbb{D} | d \succeq d_{\min}\}$ into $[0,1)$ may be used, that maps $d_{\min}$ to 0.

All the four transformations preserve non-Zeno and Zeno words.

6.3. *Reductions*

Theorems 4.1 and 5.1 may be easily lifted to the case of ω-words. In the constructions, any auxiliary state in $Q' \setminus Q$ should be assigned a rank inessential for acceptance, for instance the greatest one.

Theorem 6.2. Theorems 4.1 and 5.1 are still valid for ω-words. Both constructions preserve the type of acceptance condition.

Now we sketch adaptations of Lemmas 4.4 and 5.3. It turns out that a data or timed ω-word may fail to be a braid for precisely the same reasons as a finite one may fail, independently whether the word is non-Zeno or Zeno. Thus to check that a word is not a braid it is sufficient to inspect the finitary conditions listed in the proofs of Lemmas 4.4 and 5.3, which can be done using co-safety condition:

Lemma 6.3. In both non-Zeno and unrestricted setting, the complement of the language of data/timed braids is recognized by a nondeterministic co-safety 1-register/clock automaton.

As a consequence, the language of data/timed braids is recognized by a co-nondeterministic safety 1-register/clock automaton.

The notions of time/data isomorphism and of useless positions in a braid extend naturally to ω-words. Thus we conclude:

Lemma 6.4. In both non-Zeno and unrestricted setting, the analogues of Lemmas 4.7 and 5.6 hold for infinite words.

As all the four transformations mentioned in Section 6.2 preserve non-Zeno words, we may apply Theorem 6.2, together with Lemmas 6.3 and 6.4, to deduce the reductions:

Theorem 6.5. In both non-Zeno and unrestricted setting, the inclusion problems $\mathfrak{A} \subseteq \mathfrak{B}$ for timed and register automata are mutually inter-reducible, whenever any of the following conditions hold:

— class $\mathfrak{B}$ contains nondeterministic co-safety 1-clock/register automata and is effectively closed under union, or
— class $\mathfrak{A}$ contains co-nondeterministic safety 1-clock/register automata and is effectively closed under intersection.

7. Applications

Here we provide some evidence that the tight relationship between register automata and timed automata may be useful: we transfer a couple of results from timed do data setting.

7.1. *Finite words*

First, from the fact that 1-clock alternating timed automata have decidable emptiness (Lasota and Walukiewicz, 2008), applying Theorem 5.7 we obtain:

Theorem 7.1. The emptiness and inclusion problems for alternating 1-register automata are decidable.

Proof. Let $\mathfrak{A}$ be the class of alternating 1-clock automata. By (Lasota and Walukiewicz, 2008, Corollary 3.2), the restricted inclusion problem $\mathfrak{A} \subseteq \mathfrak{A}$ is decidable. Hence, by Theorem 5.7, the restricted inclusion problem $\mathfrak{B} \subseteq \mathfrak{B}$ is also decidable, where $\mathfrak{B}$ is the class of alternating 1-register automata. In particular, emptiness for $\mathfrak{B}$ is decidable. □

Note that register automata, as we define it here, work over *ordered* data domains and are capable of comparing data values w.r.t. $\preceq$. According to our terminology, decidability was only known for the subclass of order-blind automata (Demri and Lazić, 2009). Interestingly, the results holds for *any* total order over data.

A note on complexity. Note that alternating 1-register automata over an ordered domain have the same complexity (modulo an exponential time reduction) as alternating 1-clock timed automata. However, we must remark that these automata have a much higher complexity than *order blind* alternating 1-register automata, although both are beyond the primitive recursive functions. While the latter can be roughly bounded by the Ackermann function applied to the number of states, the complexity of the former majorizes every multiply-recursive function (in particular, Ackermann's).

More precisely, the emptiness problem for alternating timed automata with 1 clock sits in the class $\mathfrak{F}_{\omega^\omega}$ in the Fast Growing Hierarchy (Löb and Wainer, 1970)—an extension of the Grzegorczyk Hierarchy for non-primitive recursive functions—by a reduction to Lossy Channel Machines (Abdulla et al., 2005), which are known to be 'complete' for this class, i.e. in $\mathfrak{F}_{\omega^\omega} \setminus \mathfrak{F}_{<\omega^\omega}$ (Chambart and Schnoebelen, 2008). However, the emptiness problem for *order blind* alternating 1 register automata belongs to $\mathfrak{F}_\omega$ in the hierarchy, by a reduction to Incrementing Counter Automata (Demri and Lazić, 2009), which are complete for $\mathfrak{F}_\omega$ (Schnoebelen, 2010; Figueira et al., 2011). We then obtain the following result.

Corollary 7.2. The emptiness problem of alternating 1-register automata over a linearly ordered domain is in $\mathfrak{F}_{\omega^\omega} \setminus \mathfrak{F}_{<\omega^\omega}$ in the Fast Growing Hierarchy.

We show another example of a result that can be directly copied from the timed to the data setting:

Theorem 7.3. The restricted inclusion problem $\mathfrak{A} \subseteq \mathfrak{B}$, for the class $\mathfrak{A}$ of nondeterministic register automata and the class $\mathfrak{B}$ of alternating 1-register automata, is decidable and of non-primitive recursive complexity.

Proof. The analogous problem for timed automata is shown to be decidable in (Lasota and Walukiewicz, 2008), and it has a non-primitive recursive complexity since the the universality problem for alternating one clock automata is non-primitive recursive (Lasota and Walukiewicz, 2008, Corollary 4.2). We apply Theorem 5.7 since the second condition holds, obtaining that the analogous problem for register automata is decidable. We obtain

the lower bound by reduction from the problem on timed automata using Theorem 4.8. □

As the last example of application in the case of finite words, we consider the emptiness problem over a restricted class of data words. We say that a data word $(a_1, d_1) \dots (a_n, d_n)$ is *m-decreasing* iff there are at most $m - 1$ positions i with $d_i \succeq d_{i+1}$.

Theorem 7.4. Let m be a fixed non-negative number. The non-emptiness problem for alternating register automata over m-decreasing data words is decidable and non-elementary.

Proof. Again, the result is an immediate consequence of the result of (Jenkins et al., 2010, Theorems 15 and 19): emptiness of alternating timed automata over m-bounded timed words is decidable and non-elementary, where m-bounded words are those with all time stamps smaller than m.

Notice that a data word w is m-decreasing iff $\mathrm{tb}(w)$ is m-bounded. Hence, the construction of Theorem 5.1 applied to a register automaton $\mathcal{A}$ returns a timed automaton $\mathcal{B}$ such that $\mathcal{A}$ accepts an m-decreasing data word w iff $\mathcal{B}$ accepts an m-bounded timed word $\mathrm{tb}(w)$. The same arguments of Theorem 5.7 can be applied to show that the emptiness problem on alternating register automata on m-decreasing data words reduces to the emptiness problem of alternating register automata on m-bounded timed words.

Conversely, the lower bound follows from Theorems 4.1 and 4.8, and from the symmetric fact: a timed word is m-bounded iff $\mathrm{db}(w)$ is m-decreasing. □

Finally, note that the upper (decidability) bounds of Theorems 7.3 and 7.4 also apply to the class of order-blind register automata.

7.2. *Infinite words*

We will transfer results of (Parys and Walukiewicz, 2009; Ouaknine and Worrell, 2005) to register automata that give a tight decidability border in the hierarchy of weak alternating automata.

Theorem 7.5. In both non-Zeno and unrestricted setting, the emptiness problem for alternating 1-register automata with weak $(0, 1)$-parity condition is decidable.

Proof. We proceed exactly as in the proof of Theorem 7.1. The class of automata includes co-nondeterministic safety 1-register automata and is effectively closed under intersection. Thus we can apply Theorem 6.5 to obtain a reduction to the analogous problem for timed automata. The latter has been shown decidable in (Parys and Walukiewicz, 2009, Theorem 1) in the non-Zeno setting. The proof may be adapted to the unrestricted setting Parys and Walukiewicz (2011). □

Following (Parys and Walukiewicz, 2009), we argue below that the above decidability result is optimal for the weak hierarchy of alternating 1-register automata.

Theorem 7.6. In both non-Zeno and unrestricted setting, the universality of nondeterministic 1-register automata with weak $(0,1)$-parity condition is undecidable. As a consequence, emptiness of co-nondeterministic 1-register automata with weak $(1,2)$-parity condition is undecidable.

Proof. The class of automata includes nondeterministic co-safety 1-clock automata and is effectively closed under union. Thus we can apply Theorem 6.5 for a reduction from the analogous universality problem for timed automata, whose undecidability in the non-Zeno setting follows from undecidability of MTL shown in (Ouaknine and Worrell, 2005), see also (Parys and Walukiewicz, 2009, Theorem 2). The non-Zeno setting reduces easily to the unrestricted one as nondeterministic 1-register/clock automata with weak $(0,1)$-condition can easily recognize Zeno words. □

A special attention has been recently attracted (Ouaknine and Worrell, 2006; Lazić, 2008) by the class of automata that, in our terminology, corresponds to alternating safety 1-register/clock automata. From this works we obtain the following.

Theorem 7.7. In the non-Zeno setting, the emptiness problem for alternating safety 1-register automata is non-elementary.

Proof. Due to (Bouyer et al., 2008) satisfiability of safety MTL is non-elementary, and by the reduction given in (Ouaknine and Worrell, 2006) the lower bound applies to the emptiness of alternating safety 1-clock automata in the non-Zeno setting. By Theorem 6.5, the latter problem reduces to the emptiness of safety 1-register automata over non-Zeno words. □

Remark 7.8. We can not derive the above result for the unrestricted setting as (Ouaknine and Worrell, 2006) restricts to the non-Zeno setting only. Theorem 7.7 is however still interesting, when confronted with the result of (Lazić, 2008): the emptiness for order-blind safety 1-register automata is ExpSpace-complete in the unrestricted setting. The difference is due to the order on data values and the restriction to non-Zeno words.

Theorem 7.9. In the unrestricted setting, the emptiness problem for alternating safety 1-clock automata is ExpSpace-hard.

Proof. Directly by the result of (Lazić, 2008) and Theorem 6.5. □

At last, we point out two inherent restrictions of Theorem 6.5. First, note that the ExpSpace upper bound result of (Lazić, 2008) can not be transferred to timed automata unless the result still holds for ordered data. As a second example, consider the emptiness problem of order-blind alternating co-safety 1-register automata, shown non-primitive recursive in (Lazić, 2008). One would be tempted to deduce the same lower bound for the class of co-safety 1-clock automata[‡]. However, one could apply Theorem 6.5 only if this class is sufficiently expressive to recognize the language of ω-braids, for instance if it includes co-nondeterministic safety 1-clock automata. This is apparently not the case.

[‡] This lower bound can be obtained by a direct reduction from the emptiness problem for alternating 1-clock automata over finite timed words.

8. Discussion

We have shown that timed and register automata on finite and infinite words are essentially equivalent. In order to relate these two models we introduced the notion of a 'braid'-like structure, that corresponds naturally to the way a clock works when running over a timed word. As shown, most decision problems are actually equivalent for these two models. This work can be useful to derive results on one model of automata as corollaries of results on the other model, by exploiting the duality between timed and data automata shown here. This is evidenced here by showing some new results (Section 7). One limitation of our results is the both translations between timed and register automata suffer of an exponential blow-up in the size of the automaton. As a consequence, one should not expect applicability to low-complexity problems, like non-emptiness of nondeterministic timed or register automata, in both cases a PSPACE-complete problem.

As possible next step could be to investigate extended models. For instance, it is quite plausible that along the same lines one may relate timed automata with ϵ-moves on one side, and register automata with guessing (load into a register a datum chosen in a nondeterministic way) on the other.

Furthermore, a similar comparison of logical formalisms may be attempted, namely of freeze LTL and MTL or TPTL, both over finite and ω-words. This should allow to compare or transfer the complexity result for syntactic fragments of freeze LTL and MTL that appeared recently in the literature, see e.g. (Parys and Walukiewicz, 2009; Ouaknine and Worrell, 2006; Lazić, 2006; Figueira and Segoufin, 2009).

References

Parosh Aziz Abdulla, Johann Deneux, Joël Ouaknine, and James Worrell. Decidability and complexity results for timed automata via channel machines. In *ICALP*, pages 1089–1101, 2005.

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

Michael Benedikt, Clemens Ley, and Gabriele Puppis. What you must remember when processing data words. In *AMW*, 2010.

Mikołaj Bojańczyk and Sławomir Lasota. A machine-independent characterization of timed languages. In *Proc. ICALP'12*, volume 7392 of *LNCS*, pages 92–103, 2012.

Mikołaj Bojańczyk, Bartek Klin, and Sławomir Lasota. Automata with group actions. In *Proc. LICS'11*, pages 355–364, 2011.

Patricia Bouyer, Antoine Petit, and Denis Thérien. An algebraic approach to data languages and timed languages. *Inf. Comput.*, 182(2):137–162, 2003.

Patricia Bouyer, Nicolas Markey, Joël Ouaknine, Philippe Schnoebelen, and James Worrell. On termination for faulty channel machines. In *STACS*, pages 121–132, 2008.

Pierre Chambart and Philippe Schnoebelen. The ordinal recursive complexity of lossy channel systems. In *LICS*, pages 205–216. IEEE Computer Society Press, 2008. . URL `http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/CS-lics08.pdf`.

Stéphane Demri and Ranko Lazić. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3), 2009.

Diego Figueira and Luc Segoufin. Future-looking logics on data words and trees. In *MFCS*, pages 331–343, 2009.

Diego Figueira, Piotr Hofman, and Sławomir Lasota. Relating timed and register automata. In *EXPRESS*, 2010.

Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermannian and primitive-recursive bounds with Dickson's lemma. In *LICS*. IEEE Computer Society Press, 2011. To appear.

Mark Jenkins, Joël Ouaknine, Alexander Rabinovich, and James Worrell. Alternating timed automata over bounded time. In *LICS*. IEEE Computer Society Press, 2010.

Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

Sławomir Lasota and Igor Walukiewicz. Alternating timed automata. In *FoSSaCS*, pages 250–265, 2005.

Sławomir Lasota and Igor Walukiewicz. Alternating timed automata. *ACM Trans. Comput. Log.*, 9(2), 2008.

Ranko Lazić. Safely freezing LTL. In *FSTTCS*, pages 381–392, 2006.

Ranko Lazić. Safety alternating automata on data words. *CoRR*, abs/0802.4237, 2008.

M.H. Löb and S.S. Wainer. Hierarchies of number theoretic functions, I. *Archiv für Mathematische Logik und Grundlagenforschung*, 13:39–51, 1970.

Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.

Joël Ouaknine and James Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *LICS*, pages 54–63. IEEE Computer Society Press, 2004.

Joël Ouaknine and James Worrell. On the decidability of metric temporal logic. In *LICS*, pages 188–197. IEEE Computer Society Press, 2005.

Joël Ouaknine and James Worrell. Safety metric temporal logic is fully decidable. In *TACAS*, pages 411–425, 2006.

Paweł Parys and Igor Walukiewicz. Weak alternating timed automata. In *ICALP*, pages 273–284, 2009.

Paweł Parys and Igor Walukiewicz. Personal communication, 2011.

Philippe Schnoebelen. Revisiting Ackermann-hardness for lossy counter systems and reset Petri nets. In *MFCS 2010*, volume 6281 of *LNCS*. Springer, 2010.