Alternating Timed Automata *

Sławomir Lasota[1]** and Igor Walukiewicz[2]

[1] Institute of Informatics, Warsaw University
Banacha 2, 02-097 Warszawa

[2] LaBRI, Université Bordeaux-1
351, Cours de la Libération, F-33 405, Talence cedex, France

Abstract. A notion of alternating timed automata is proposed. It is shown that such automata with only one clock have decidable emptiness problem over finite words. This gives a new class of timed languages which is closed under boolean operations and which has an effective presentation. We prove that the complexity of the emptiness problem for alternating timed automata with one clock is non-primitive recursive. The proof gives also the same lower bound for the universality problem for nondeterministic timed automata with one clock. Over infinite words, we show undecidability of the universality problem.

1 Introduction

Timed automata is a widely studied model of real-time systems. It is obtained from finite nondeterministic automata by adding clocks which can be reset and whose values can be compared with constants. In this paper we consider alternating version of timed automata obtained by introducing universal transitions in the same way as it is done for standard nondeterministic automata. From the results of Alur and Dill [3] it follows that such a model cannot have decidable emptiness problem as the universality problem for timed automata is not decidable. In the recent paper [20] Ouaknine and Worrell has shown that the universality problem is decidable for nondeterministic automata with one clock, over finite timed words. Inspired by their construction, we show that the emptiness problem for alternating timed automata with one clock is decidable as well. We also prove not primitive recursive lower bound for the problem. The proof implies the same bound for the universality problem for nondeterministic timed automata with one clock, thereby answering the question posed by Ouaknine and Worrell [20]. To complete the picture we also show that an extension of our model with epsilon-transitions has undecidable emptiness problem. Furthermore, we prove undecidability of the universality problem for one-clock nondeterministic automata over infinite timed words.

* Work reported here has been partially supported by the European Community Research Training Network Games.

** Partially supported by the Polish Kbn grant No. 4 T11C 042 25. This work was performed in part during the author's stay at LaBRI, Université Bordeaux-1.

The crucial property of timed automata models is the decidability of the emptiness problem. The drawback of the model is that the class of languages recognized by timed automata is not closed under complement and the universality question is undecidable (Π_1^1-hard) [3]. One solution to this problem is to restrict to deterministic timed automata. Another, is to restrict the reset operation; this gives the event-clock automata model [5]. A different ad-hoc solution could be to take the boolean closure of the languages recognized by timed automata. This solution does not seem promising due to the complexity of the universality problem. This consideration leads to the idea of using automata with one clock for which the universality problem is decidable. The obtained class of alternating timed automata is by definition closed under boolean operations. Moreover, using the method of Ouaknine and Worrell, we can show that the class has decidable emptiness problem. As it can be expected, there are languages recognizable by timed automata that are not recognizable by alternating timed automata with one clock. More interestingly, the converse is also true: there are languages recognizable by alternating timed automata with one clock that are not recognizable by nondeterministic timed automata with any number of clocks.

Once the decidability of the emptiness problem for alternating timed automata with one clock is shown, the next natural question is the complexity of the problem. We show a non-primitive recursive lower bound. For this we give a reduction of the reachability problem for lossy channel systems [22]. The reduction shows that the lower bound holds also for purely universal alternating timed automata. This implies non-primitive recursive lower bound for the universality problem for nondeterministic timed automata with one clock. We also point out that allowing epsilon transitions in our model permits to code perfect channel systems and hence makes the emptiness problem undecidable.

All this applies to automata over finite timed words. In the case of infinite words, we prove undecidability of the universality problem of nondeterministic automata with one clock, by the reduction of the halting problem. This immediately implies undecidability of the emptiness problem for alternating one-clock automata.

Related work Our work is strongly inspired by the results of Ouaknine and Worrell [20]. Except for [13], it seems that the notion of alternation in the context of timed automata was not studied before. The reason was probably undecidability of the universality problem. The alternating automata introduced in [13] run over infinite timed trees and were used to show decidability of model checking for TCTL. Emptiness for these automata is apparently undecidable, even under one-clock restriction, in view of our result for one-clock automata over infinite words. On the other hand, emptiness for nondeterministic timed tree automata is decidable [18].

Some research (see [7, 12, 9, 4, 8] and references within) was devoted to the control problem in the timed case. While in this case one also needs to deal with some universal branching, these works do not seem to have direct connection to our setting.

Furthermore, let us mention that restrictions to one clock (and two clocks) have been already considered in the context of TCTL model-checking of timed systems [14, 19], leading to a lower complexity in some cases. Finally, in [6] the parametric variant of emptiness problem was shown decidable under restriction to one clock (similarly as in our setting) and undecidable for three clocks; the two-clock case was left as an open question.

Similar results to ours were obtained independently by Ouaknine and Worrell [21] and by Abdulla et al [2]. The former paper defines alternating timed automata, in a slightly different way than ours, and applies these automata to prove decidability of model-checking for Metric Temporal Logic. The non-primitive recursive lower bound is also established. In the latter paper, the undecidability result for the universality problem over infinite words is reported.

Organization of the paper In the next section we define alternating timed automata; we discuss their basic properties and relations with nondeterministic timed automata. In Section 3 we show decidability of the emptiness problem for alternating timed automata with one clock. In the following two section we show a non-primitive recursive lower bound for the problem, and then the undecidability result for an extension of our model with epsilon-moves. In Section 6 we investigate automata over infinite words.

A preliminary version of this article appeared as [17].

2 Alternating Timed Automata

In this section we introduce the alternating timed automata model and study its basic properties. The model is a quite straightforward extension of the nondeterministic model. Nevertheless some care is needed to have the desirable feature that complementation corresponds to exchanging existential and universal branchings (and final and non-final states). As can be expected, alternating timed automata can recognize more languages than their nondeterministic counterparts. The price to pay for this is that the emptiness problem becomes undecidable, in contrast to timed automata [3]. This motivates the restriction to automata with one clock. With one clock alternating automata can still recognize languages not recognizable by nondeterministic automata and moreover, as we show in the next section, they have decidable emptiness problem.

For a given finite set $\mathcal{C}$ of *clock variables* (or *clocks* in short), consider the set $\Phi(\mathcal{C})$ of clock constraints σ defined by

$$\sigma \quad ::= \quad x < c \;\mid\; x \leq c \;\mid\; \sigma_1 \wedge \sigma_2 \;\mid\; \neg\sigma,$$

where c stands for an arbitrary nonnegative integer constant, and $x \in \mathcal{C}$. For instance, note that tt (always true), or $x = c$, can be defined as abbreviations. Each constraint σ denotes a subset $[\sigma]$ of $(\mathbb{R}_+)^{\mathcal{C}}$, in a natural way, where $\mathbb{R}_+$ stands for the set of nonnegative reals.

Transition relation of a timed automaton [3] is usually defined by a finite set of rules δ of the form

$$\delta \subseteq Q \times \Sigma \times \Phi(\mathcal{C}) \times Q \times \mathcal{P}(\mathcal{C}),$$

where Q is a set of *locations* (control states) and Σ is an input alphabet. A rule $\langle q, a, \sigma, q', r\rangle \in \delta$ means, roughly, that when in a location q, if the next input letter is a and the constraint σ is satisfied by the current valuation of clock variables, the next location can be q' and the clocks in r should be reset to 0. Our definition below uses an easy observation, that the relation δ can be suitably rearranged into a finite partial function

$$Q \times \Sigma \times \Phi(\mathcal{C}) \dot{\to} \mathcal{P}(Q \times \mathcal{P}(\mathcal{C})).$$

The definition below comes naturally when one thinks of an element of the codomain as a disjunction of a finite number of pairs (q, r). Let $\mathcal{B}^+(X)$ denote the set of all positive boolean formulas over the set X of propositions, i.e., the set generated by:

$$\phi \quad ::= \quad X \ \mid \ \phi_1 \wedge \phi_2 \ \mid \ \phi_1 \vee \phi_2.$$

Definition 1 (Alternating timed automaton). *An* alternating timed automaton *is a tuple $\mathcal{A} = (Q, q_0, \Sigma, \mathcal{C}, F, \delta)$ where: Q is a finite set of locations, Σ is a finite input alphabet, $\mathcal{C}$ is a finite set of clock variables, and $\delta : Q \times \Sigma \times \Phi(\mathcal{C}) \dot{\to} \mathcal{B}^+(Q \times \mathcal{P}(\mathcal{C}))$ is a finite partial function. Moreover $q_0 \in Q$ is an* initial state *and $F \subseteq Q$ is a set of* accepting states. *We also put an additional restriction:*

(Partition) *For every q and a, the set $\{[\sigma] : \delta(q, a, \sigma)$ is defined$\}$ gives a (finite) partition of $(\mathbb{R}_+)^{\mathcal{C}}$.*

The (Partition) condition does not limit the expressive power of automata. We impose it because it permits to give a nice symmetric semantic for the automata as explained below. We will often write rules of the automaton in a form: $q, a, \sigma \ \mapsto \ b$.

By a *timed word* over Σ we mean a finite sequence

$$w = (a_1, t_1)(a_2, t_2) \dots (a_n, t_n) \tag{1}$$

of pairs from $\Sigma \times \mathbb{R}_+$. Each t_i describes the amount of time that passed between reading a_{i-1} and a_i, i.e., a_1 was read at time t_1, a_2 was read at time t_1+t_2, and so on. In Sections 4 and 5 it will be more convenient to use an alternative representation where t_i denotes the time elapsed since the beginning of the word.

To define an execution of an automaton, we will need two operations on valuations $\mathbf{v} \in (\mathbb{R}_+)^{\mathcal{C}}$. A valuation $\mathbf{v}+t$, for $t \in \mathbb{R}_+$, is obtained from $\mathbf{v}$ by augmenting value of each clock by t. A valuation $\mathbf{v}[r := 0]$, for $r \subseteq \mathcal{C}$, is obtained by reseting values of all clocks in r to zero.

For an alternating timed automaton $\mathcal{A}$ and a timed word w as in (1), we define the *acceptance game* $G_{\mathcal{A},w}$ between two players Adam and Eve. Intuitively, the objective of Eve is to accept w, while the aim of Adam is the opposite. A play starts at the initial configuration $(q_0, \mathbf{v}_0)$, where $\mathbf{v}_0 : \mathcal{C} \to \mathbb{R}_+$ is a valuation assigning 0 to each clock variable. It consists of n phases. The $(k{+}1)$-th phase starts in $(q_k, \mathbf{v}_k)$, ends in some configuration $(q_{k+1}, \mathbf{v}_{k+1})$ and proceeds as follows. Let $\bar{\mathbf{v}} := \mathbf{v}_k + t_{k+1}$. Let σ be the unique constraint such that $\bar{\mathbf{v}}$ satisfies σ and $b = \delta(q_k, a_{k+1}, \sigma)$ is defined. Now the outcome of the phase is determined by the formula b. There are three cases:

- $b = b_1 \wedge b_2$: Adam chooses one of subformulas b_1, b_2 and the play continues with b replaced by the chosen subformula;
- $b = b_1 \vee b_2$: dually, Eve chooses one of subformulas;
- $b = (q, r) \in Q \times \mathcal{P}(\mathcal{C})$: the phase ends with the result $(q_{k+1}, \mathbf{v}_{k+1}) := (q, \bar{\mathbf{v}}[r := 0])$. A new phase is starting from this configuration if $k{+}1 < n$.

The winner is Eve if q_n is accepting $(q_n \in F)$, otherwise Adam wins.

Definition 2 (Acceptance). *The automaton $\mathcal{A}$* accepts w *iff Eve has a winning strategy in the game $G_{\mathcal{A},w}$. By $L(\mathcal{A})$ we denote the language of all timed words w accepted by $\mathcal{A}$.*

To show the power of alternation we give an example of an automaton for a language not recognizable by standard (i.e. nondeterministic) timed automata (cf. [3]).

Example 1. Consider a language consisting of timed words w over a singleton alphabet $\{a\}$ that contain no pair of letters such that one of them is precisely one time unit later than the other. The alternating automaton for this language has three states q_0, q_1, q_2. State q_0 is initial. The automaton has a single clock x and the following transition rules:

$$q_0, a, tt \mapsto (q_0, \emptyset) \wedge (q_1, \{x\}) \qquad q_1, a, x{\neq}1 \mapsto (q_1, \emptyset)$$
$$q_1, a, x{=}1 \mapsto (q_2, \emptyset) \qquad q_2, a, tt \mapsto (q_2, \emptyset)$$

States q_0 and q_1 are accepting, q_2 is not. In state q_0, at each input letter, Adam chooses either to stay in q_0 either to to go to q_1; In the latter case clock x is reset. Furthermore, the automaton can only quit state q_1 exactly one time unit after entering it. Hence, Adam has a strategy to reach q_2 iff the word is not in the language, i.e., some letter is one time unit after some other.

As one expects, we have the following:

Proposition 1. *The class of languages accepted by alternating timed automata is effectively closed under all boolean operations: union, intersection and complementation. These operations to do not increase the number of clocks of the automaton.*

The closure under conjunction and disjunction is straightforward since we permit positive boolean expressions as values of the transition function. Due to the condition (Partition) the automaton $\neg\mathcal{A}$ for the complement is obtained from $\mathcal{A}$ by exchanging conjunctions with disjunctions in all transitions and exchanging accepting states with non-accepting states.

Definition 3. *An alternating timed automaton $\mathcal{A}$ is called* purely universal *if the disjunction does not appear in the transition rules δ. Dually, $\mathcal{A}$ is* purely existential *if no conjunction appears in δ.*

Clearly, if $\mathcal{A}$ is purely universal (purely existential) then $\neg\mathcal{A}$ is purely existential (purely universal). It is obvious that every purely-existential automaton is a standard nondeterministic timed automaton. The converse requires a proof because of the (Partition) condition.

Proposition 2. *Every standard nondeterministic automaton is equivalent to a purely-existential automaton.*

Proof. Transition relation of a nondeterministic timed automaton is usually defined by a finite set δ of rules of the form $\langle q, a, \sigma, q', r\rangle \in Q \times \Sigma \times \Phi(\mathcal{C}) \times Q \times \mathcal{P}(\mathcal{C})$. Given such an automaton $\mathcal{A}$, the corresponding purely- existential alternating automaton $\widehat{\mathcal{A}}$ has the same set Q of states as $\mathcal{A}$, plus one additional state $q_{\mathtt{sink}}$. Automaton $\widehat{\mathcal{A}}$ has the same initial state and accepting states as $\mathcal{A}$, the same set of clocks $\mathcal{C}$, and the same input alphabet. The only essential difference is that δ is replaced by $\widehat{\delta} : Q \times \Sigma \times \Phi(\mathcal{C}) \dot{\to} \mathcal{B}^+(Q \times \mathcal{P}(\mathcal{C}))$, defined as follows.

In fact, we prefer to define $\widehat{\delta}$ equivalently as $\widehat{\delta} : Q \times \Sigma \times \Phi(\mathcal{C}) \dot{\to} \mathcal{P}(Q \times \mathcal{P}(\mathcal{C}))$. Let $\sigma_1 \ldots \sigma_n$ be all clock constraints appearing in δ. The guards appearing in $\widehat{\delta}$ will be σ_X, for $X \subseteq \{1 \ldots n\}$, defined by:

$$\sigma_X = \wedge_{i \in X} \sigma_i \quad \wedge \quad \wedge_{i \notin X} \neg\sigma_i.$$

I.e., we consider conjunctions of arbitrary sets of guards σ_i. The value $\widehat{\delta}(q, a, \sigma)$ is defined iff $\sigma = \sigma_X$ for some X, hence $\widehat{\delta}$ clearly satisfies the (Partition) condition. The constraints σ_X satisfying $[\sigma_X] = \emptyset$ can be safely omitted. We put:

$$\widehat{\delta}(q, a, \sigma_X) = \{(q', r) : \langle q, a, \sigma_i, q', r\rangle \in \delta \;\text{ for some }\; i \in X\}.$$

If $\widehat{\delta}(q, a, \sigma_X)$ is empty, we put $\widehat{\delta}(q, a, \sigma_X) = \{(q_{\mathtt{sink}}, \emptyset)\}$. And finally we put: $\widehat{\delta}(q_{\mathtt{sink}}, a, \sigma_X) = \{(q_{\mathtt{sink}}, \emptyset)\}$, for any a and σ_X.

It is routine now to check that languages accepted by $\mathcal{A}$ and $\widehat{\mathcal{A}}$ coincide. $\square$

In the following sections, we consider emptiness, universality and containment for different classes of alternating timed automata. For clarity, we recall definitions here.

Definition 4. *For a class C of automata we consider three problems:*

- *Emptiness: given $\mathcal{A} \in C$ is $L(\mathcal{A})$ empty.*
- *Universality: given $\mathcal{A} \in C$ does $L(\mathcal{A})$ contain all timed words.*
- *Containment: given $\mathcal{A}, \mathcal{B} \in C$ does $L(\mathcal{A}) \subseteq L(\mathcal{B})$.*

It is well known that the universality is undecidable for non-deterministic timed automata [3] with at least two clocks. As a consequence, two other problems are also undecidable for alternating timed automata with two clocks. This is why in the rest of the paper we focus on automata with one clock only.

Proviso: In the following all automata have one clock.

The automaton from Example 1 uses only one clock. This shows that one clock alternating automata can recognize some languages not recognizable by nondeterministic automata with many clocks. The converse is also true:

Theorem 1. *Classes of languages recognizable by nondeterministic timed automata and by one-clock alternating timed automata are incomparable.*

Proof. We show a language acceptable by a nondeterministic automaton with many clocks but not acceptable by an alternating automaton with one clock.

Consider the timed language over the singleton alphabet $\{b\}$ consisting of the words containing appearances of the letter b at times t_1 and t_2, where $0 < t_1 < t_2 < 1$, no other b in between 0 and 1 and precisely one b between $t_1 + 1$ and $t_2 + 1$. We will show that this language cannot be accepted by an alternating timed automaton with one clock. Obviously it is accepted by a nondeterministic timed automaton with two clocks.

For a preparation consider a deterministic untimed automaton $\mathcal{B}$. A sequence b^k of k letters b determines a function $f_k^{\mathcal{B}} : Q^{\mathcal{B}} \to Q^{\mathcal{B}}$ saying that if started in the state q after reading b^k the automaton will end in $f_k^{\mathcal{B}}(q)$. Clearly the number of such functions is bounded if the number of states is fixed. Thus there are m and l, depending only on the number of states, such that $f_m^{\mathcal{B}} = f_{m+l}^{\mathcal{B}}$. Moreover $f_{m+i}^{\mathcal{B}} = f_{m+l+i}^{\mathcal{B}}$ for all $i > 0$.

To arrive at a contradiction assume that our language is recognized by an ATA $\mathcal{A}$ with n states. Suppose for a moment that all constants in the tests in transition function of the automaton are integers. Let m and l be such that $f_{m+i}^{\mathcal{B}} = f_{m+l+i}^{\mathcal{B}}$ for all $i > 0$ and for all deterministic automata $\mathcal{B}$ with at most $2^{2^{2n}}$ states.

Now consider two words w_1 and w_2. In w_1 we have b at times 0.3, 0.7, 1.5 and m b's somewhere in the interval $(1, 1.3)$ as well as m b's somewhere in the interval $(1.7, 2)$. Word w_2 is obtained from w_1 by adding l b's somewhere in the interval $(1.3, 1.7)$; but not at point 1.5 of course. We will show that if $\mathcal{A}$ accepts w_1 then it also accepts w_2.

Consider the accepting run of $\mathcal{A}$ on w_1. Look at the configurations in which the automaton reaches at time 1. Let (q, v) be one of them. The value of the clock v can be 0.3, 0.7 or 1. This is because there are only two letters till 1 and the automaton can reset clock only when it reads a letter. We will analyse the three cases one by one.

If $v = 1$ then it is easy to see that from a configuration (q, v) the automaton has no use for the clock in the interval $(1, 2)$. If not reset, the value of the clock in this interval will be in $(1, 2)$ and the automaton can compare the values only with integers. If the clock is reset then its value will stay in $(0, 1)$ till the end of the interval. Thus from the configuration (q, v) automaton $\mathcal{A}$ behaves as an alternating automaton without a clock with additional flag telling whether there was a reset or not. Because it has n states, it is equivalent to a deterministic automaton of at most $2^{2^{2n}}$ states. We have that if it accepts from q the string of $2m + 1$ letters b then it also accepts $2m + l + 1$ letters b. Thus $\mathcal{A}$ has an accepting run from (q, v) in w_2 if it had one in w_1.

If $v = 0.7$ then consider the run of $\mathcal{A}$ from (q, v) till the time point 1.3. Automaton $\mathcal{A}$ has no use of the clock till that point for the same reason as above. It arrives at a set of configurations: some with the value of the clock 1 and some with the value < 0.3. The later are possible because $\mathcal{A}$ could reset a clock. Consider the rest of the computation starting from a configuration $(q', 1)$. Once again the clock will not be useful to $\mathcal{A}$ in the rest of the word. Hence we will arrive to the same final states on a^{1+m} and a^{1+m+l}. Similarly for all the configurations with the values of the clock $< .3$.

If $v = 0.3$ then consider the run of $\mathcal{A}$ from (q, v) till the time point 1.7. Till that time there was no use of the clock. We get a set of configurations with clock value 1 and the other with clock value < 0.7. The possible configurations with clock value 1 are the same no matter if we have made automaton run on w_1 or on w_2, for the same reason as before. As the rest of w_1 is the same as the rest of w_2 we are done. On the other hand, when comparing configurations with clock value < 0.7 in runs over w_1 and w_2, the possible locations are the same but the clock values may differ. But the clock value is irrelevant before time 2, hence again we are done.

In the argument we essentially use the assumption that we compare clocks only with natural numbers. If we allowed to compare with rationals we can get an example of the similar kind by using rescaling. Instead of intervals $(0, 1)$ and $(1, 2)$ we would use smaller intervals that are of the size smaller than the smallest constant used by the automaton.

More precisely, let $c \neq 0$ be the smallest positive rational such that the clock is compared in $\mathcal{A}$ either to c or to $1-c$ or to $1+c$. We define words w_1 and w_2 as follows. In w_1 we have b at times $0.3c$, $0.7c$, $1 + 0.5c$ and m b's somewhere in the interval $(1, 1+0.3c)$ as well as m b's somewhere in the interval $(1+0.7c, 1+c)$. Word w_2 is obtained from w_1 by adding l b's somewhere in the interval $(1 + 0.3c, 1 + 0.7c)$; but not at point $1 + 0.5c$. The whole proof works unchanged. $\square$

3 Decidability

The main result of this section is that the emptiness problem for one-clock alternating timed automata is decidable. Due to closure under boolean operations, this implies the decidability of the universality and the containment problems.

Theorem 2. *The emptiness problem is decidable for one-clock alternating timed automata.*

Corollary 1. *The containment problem is decidable for one-clock alternating timed automata.*

The rest of this section is devoted to the proof of Theorem 2. Essentially, we have adapted the method of Ouaknine and Worrell [20] for our more general setting. We point out the differences below.

Fix a one-clock alternating timed automaton $\mathcal{A} = (Q, q_0, \Sigma, \{x\}, F, \delta)$. For readability, assume w.l.o.g. that the boolean conditions appearing in rules of δ are all in *disjunctive normal form.* In terms of acceptance games this means that each phase consists of a single move of Eve followed by a single move of Adam. Consider a labelled transition system $\mathcal{T}$ whose states are finite sets of configurations, i.e., finite sets of pairs $(q, \mathbf{v})$, where $q \in Q$ and $\mathbf{v} \in \mathbb{R}_+$. The initial position in $\mathcal{T}$ is $P_0 = \{(q_0, \mathbf{0})\}$ and there is a transition $P \xrightarrow{a,t} P'$ in $\mathcal{T}$ iff P' can be obtained from P by the following nondeterministic process:

- First, for each $(q, \mathbf{v}) \in P$, do the following:
  - let $\mathbf{v}' := \mathbf{v}+t$,
  - let $b = \delta(q, a, \sigma)$ for the uniquely determined σ satisfied in $\mathbf{v}'$,
  - choose one of disjuncts of b, say

$$(q_1, r_1) \ \wedge \ \ldots \ \wedge \ (q_k, r_k) \qquad (k > 0),$$

  - let $\mathtt{Next}_{(q,\mathbf{v})} = \{(q_i, \mathbf{v}'[r_i := 0]) : i = 1 \ldots k\}$.
- Then, let $P' := \bigcup_{(q,\mathbf{v}) \in P} \mathtt{Next}_{(q,\mathbf{v})}$.

This construction is very similar to the translation from alternating to nondeterministic automata over (untimed) words: we just collect all universal choices in one set. Compared to [20], the essential difference is that we have to deal with both disjunction and conjunction, while in [20] only one of them appeared. We treat conjunction similarly to determinization in [20]. On the other hand, we leave the existential choice, i.e., nondeterminism, essentially unaffected in $\mathcal{T}$.

In what follows we will derive from $\mathcal{T}$ a finite-branching transition system $\mathcal{H}$, suitable for the decision procedure. Like in [20], the degree of the nodes of $\mathcal{H}$ will not be bounded but nevertheless finite. This is sufficient for our purposes.

A state $\{(q_1, \mathbf{v}_1), \ldots, (q_n, \mathbf{v}_n)\}$ of $\mathcal{T}$ is called *bad* iff all control states q_i are accepting ($q_i \in F$). The following proposition characterizes acceptance in $\mathcal{A}$ in terms of reachability of bad states in $\mathcal{T}$. It is enough to consider reachability because $\mathcal{A}$ accepts only finite words.

Lemma 1. *$\mathcal{A}$ accepts a timed word w iff there is a path in $\mathcal{T}$, labelled by w, from P_0 to a bad state.*

Let $\widehat{\mathcal{T}}$ be a labelled transition system obtained from $\mathcal{T}$ by erasing time information from transition labels, i.e., there is a transition $P \xrightarrow{a} Q$ in $\widehat{\mathcal{T}}$ iff there is $P \xrightarrow{a,t} Q$ in $\mathcal{T}$, for some $t \in \mathbb{R}_+$. Now we cannot talk about particular timed words but still we have the following:

Lemma 2. *$L(\mathcal{A})$ is nonempty if and only if there is a path in $\widehat{\mathcal{T}}$ from P_0 to a bad state.*

Thus, the (non)emptiness problem for $\mathcal{A}$ is reduced to the reachability of a bad state in $\widehat{\mathcal{T}}$. The last difficulty is that even if each state of $\widehat{\mathcal{T}}$ is a finite set, there are uncountably many states. The following definition allows to abstract from the precise timing information in states. Let $c_{\max}$ denote the biggest constant appearing in constraints in δ. Let set $\mathtt{reg}$ of *regions* be a partition of $\mathbb{R}_+$ into $2 \cdot (c_{\max}+1)$ sets as follows:

$$\mathtt{reg} := \{\{0\}, (0,1), \{1\}, (1,2), \ldots, (c_{\max}-1, c_{\max}), \{c_{\max}\}, (c_{\max}, +\infty)\}.$$

For $\mathbf{v} \in \mathbb{R}_+$, let $\mathtt{reg}(\mathbf{v})$ denote its region; and let $\mathtt{fract}(\mathbf{v})$ denote the fractional part of $\mathbf{v}$. Below we work with finite words over the alphabet $\Lambda = \mathcal{P}(Q \times \mathtt{reg})$ consisting of finite sets of pairs $(q, \mathtt{r})$, where $q \in Q$ is a control state and $\mathtt{r} \in \mathtt{reg}$ is a region.

Definition 5. *For a state P of $\widehat{\mathcal{T}}$ we define a word $H(P)$ from Λ^* as the one obtained by the following procedure:*

- *replace each $(q, \mathbf{v}) \in P$ by a triple $\langle q, \mathtt{reg}(\mathbf{v}), \mathtt{fract}(\mathbf{v})\rangle$ (this yields a finite set of triples)*
- *sort all these triples w.r.t.* $\mathtt{fract}(\mathbf{v})$ *(this yields a finite sequence of triples)*
- *group together triples that have the same value of* $\mathtt{fract}(\mathbf{v})$*, ignoring multiple occurrences (this yields a finite sequence of finite sets of triples)*
- *forget about* $\mathtt{fract}(\mathbf{v})$*, i.e., replace each triple $\langle q, \mathtt{reg}(\mathbf{v}), \mathtt{fract}(\mathbf{v})\rangle$ by a pair $(q, \mathtt{reg}(\mathbf{v}))$ (this yields a finite sequence of finite sets of pairs, a word in Λ^*).*

Definition 6. *Define an equivalence relation $\sim$ over states of $\widehat{\mathcal{T}}$ as the kernel of H, i.e., $P \sim P'$ iff $H(P) = H(P')$.*

The following observations are straightforward:

Lemma 3. *Relation $\sim$ is a bisimulation over transition system $\widehat{\mathcal{T}}$.*

Lemma 4. *If P is bad and $P \sim P'$ then P' is bad.*

Let $\mathcal{H}$ denote the quotient of the transition system $\widehat{\mathcal{T}}$ by $\sim$. To put it more explicitly: states of $\mathcal{H}$ are all words $H(P)$, for a state P of $\widehat{\mathcal{T}}$; there is a transition $W_1 \xrightarrow{a} W_2$ in $\mathcal{H}$ if there is a transition $P_1 \xrightarrow{a} P_2$ in $\widehat{\mathcal{T}}$ with $H(P_1) = W_1$, $H(P_2) = W_2$. Since $\sim$ is a bisimulation, the definition does not depend on a particular choice of P_1 (and P_2). The initial state W_0 in $\mathcal{H}$ is $H(P_0)$.

By Lemma 4 it is correct to call a state W in $\mathcal{H}$ *bad* if $W = H(P)$ for a bad state P. Because $\mathcal{H}$ is a quotient of $\widehat{\mathcal{T}}$ by bisimulation, from Lemma 2 we derive:

Lemma 5. *$L(\mathcal{A})$ is nonempty iff a bad state is reachable in $\mathcal{H}$ from W_0.*

At this point, we have reduced emptiness of $L(\mathcal{A})$ to the reachability of a bad state in a countably infinite transition system $\mathcal{H}$. The rest of the proof is quite standard [1, 15] and exploits the fact that one can put an appropriate *well-quasi-order* (*wqo* in short) on states of $\mathcal{H}$. Unfortunately, we are obliged to redo the proofs as we could not find a theorem that fits precisely our setting.

Definition 7. *Let $\preceq$ denote the* monotone domination *ordering over Λ^* induced by the subset inclusion over Λ, defined as follows: $a_1 \ldots a_n \preceq b_1 \ldots b_m$ iff there exists a strictly increasing function $f : \{1, \ldots, n\} \to \{1, \ldots, m\}$ such that for each $i \leq n$, $a_i \subseteq b_{f(i)}$.*

Lemma 6 ([16]). *Relation $\preceq$ is a wqo, i.e., for arbitrary infinite sequence $W_1, W_2, \ldots$ of words over Λ, there exist indexes $i < j$ such that $W_i \preceq W_j$.*

The decision procedure for reachability of bad states will work by an exhaustive search through a sufficiently large portion of the whole reachability tree. Thus we need to know that an arbitrarily large part of that tree can be effectively constructed. Roughly, all time delays of an action a from W can be captured by a finite number of cyclic shifts of W with an appropriate change of region.

Lemma 7. *For each state W in $\mathcal{H}$, its set of successors $\{W' \in \Lambda^* : W \xrightarrow{a} W' \text{ for some } a\}$ is finite and effectively computable.*

Proof. Recall that a word W represents a finite set of pairs $(q, \mathbf{v})$. The letters are sorted according to the value of $\mathtt{fract}(\mathbf{v})$; moreover the letters represent finite sets of pairs in fact, i.e., all the pairs with the same $\mathtt{fract}(\mathbf{v})$. Note that all pairs with $\mathtt{fract}(\mathbf{v}) = 0$, if any, are represented by by the first letter of W; and the corresponding region is of the form $\{i\}$ (or $(c_{\max}, \infty)$) in this case.

Now imagine a transition $W \xrightarrow{a} W'$ in $\mathcal{H}$. This corresponds to some transition $P \xrightarrow{a,t} P'$ in $\mathcal{T}$, for some t and some chosen set P of pairs $(q, \mathbf{v})$. Importantly, the same time delay t is applied to all the pairs $(q, \mathbf{v})$. Denote by $\widehat{P}$ the set obtained form P by time delay, i.e., by replacing each $(q, \mathbf{v})$ with $(q, \mathbf{v}+t)$; consider this, conceptually, for all $t > 0$. The corresponding word $\widehat{W}$ in $\mathcal{H}$ is obtained from W by an operation similar to a cyclic shift, to the right, repeated as many times as needed. This operation modifies W as follows. Note that the first letter of W contains either only pairs of the form $(q, \{i\})$, either only the pairs of the form $(q, (i, i+1))$ (and perhaps $(c_{\max}, \infty)$ as well). In the first case, change each region $\{i\}$ in the first letter of W to $(i, i+1)$ (or to $(c_{\max}, \infty)$, if $i = c_{\max}$). In the second case, remove the right-most letter and put it as the first letter in the word, and change each region $(i, i+1)$ to $\{i+1\}$.

Hence, the set $\{W' \in \Lambda^* : W \xrightarrow{a} W' \text{ for some } a\}$ can be computed by applying the operation defined above an arbitrary number of times (until all regions are $(c_{\max}, \infty)$), yielding $\widehat{W}$; and by calculating the effect of performing any transition a from $\widehat{W}$. $\square$

The following observation is proved in the same way as Lemma 15 in [20].

Lemma 8. *The inverse of $\preceq$ relation is a simulation: whenever $W_1 \preceq W_2$ and $W_2 \xrightarrow{a} W_2'$, there is some W_1' such that $W_1 \xrightarrow{a} W_1'$ and $W_1' \preceq W_2'$.*

Proof. We combine two simple observations. First, $H(P_1) \preceq H(P_2)$ iff $H(P_1) = H(P)$, for some $P \subseteq P_2$, i.e., $P_1 \sim P$, for some $P \subseteq P_2$. Second, whenever $P_2 \xrightarrow{a,t} P_2'$ and $P \subseteq P_2$, there exists $P' \subseteq P_2'$ with $P \xrightarrow{a,t} P'$.

Assume $W_1 \preceq W_2$; we have some P_1, P_2 and P as above, with $H(P_i) = W_i$. Assume furthermore $W_2 \xrightarrow{a} W_2'$; hence $P_2 \xrightarrow{a,t_2} P_2'$, for some t_2 such that $H(P_2') = W_2'$. It follows $P \xrightarrow{a,t_2} P'$ for some $P' \subseteq P_2'$. And, by Lemma 3, $P_1 \xrightarrow{a,t_1} P_1'$ for some t_1 and P_1' such that $H(P_1') = H(P')$. Hence, putting $W_1' = H(P_1')$, we get $W_1 \xrightarrow{a} W_1'$ and $W_1' \preceq W_2'$. $\square$

The next observation is more specific to our setting but fortunately very easy.

Lemma 9 (Downward closedness of badness). *Whenever $W \preceq W'$ and W' is bad then W is bad as well.*

Proof. Take a letter w_i of W. We need to show that $q \in F$ for every $(q, \mathtt{r}) \in w_i$. By the definition of $W \preceq W'$ we have $w_i \subseteq w'_j$ for some letter w'_j of W'. Hence, $(q, \mathtt{r}) \in w'_j$ and $q \in F$ as W' is bad. $\square$

Now we are ready to prove the main lemma.

Lemma 10. *It is decidable whether a bad state is reachable in $\mathcal{H}$ from W_0.*

Proof. The *reachability tree* is the unravelling of $\mathcal{H}$ from W_0. The algorithm constructs a portion t of the tree conforming to the following rule: do not add a node W' to t in a situation when among its ancestors there is some $W \preceq W'$. Lemma 6 guarantees that each path in t is finite. Furthermore, since the degree of each node is finite, t is a finite tree.

We need only to prove that if a bad state is reachable in $\mathcal{H}$ from W_0 then t contains at least one bad state. Let W be such a bad state reachable from W_0 in $\mathcal{H}$ by a path π of the shortest length. Assume that W is not in t, i.e., there are two other nodes in π, say W_1 and W_2 such that W_1 is an ancestor of W_2 in the reachability tree and $W_1 \preceq W_2$ (i.e., W_2 was *not* added into t). Since the inverse of $\preceq$ is a simulation by Lemma 8, the sequence of transitions in π from W_2 to W can be imitated by the corresponding sequence of transitions from W_1 to some other $W' \preceq W$. W' is bad as well by Lemma 9. Moreover, the path leading to W' is strictly shorter than π, a contradiction. $\square$

Theorem 2 follows immediately from Lemma 10 and Lemma 5.

Remark: In fact, Ouaknine and Worrell showed decidability of containment "$L(\mathcal{A}) \subseteq L(\mathcal{B})$" in a slightly more general case, namely when automaton $\mathcal{A}$ has an arbitrary many clocks. Along the same lines one can adapt our proof, assumed that $\mathcal{A}$ is an arbitrary nondeterministic timed automaton and $\mathcal{B}$ is a one-clock alternating timed automaton. We sketch below the necessary modifications.

If we denote by $\bar{\mathcal{B}}$ a dual of $\mathcal{B}$, i.e., an automaton accepting the complement of $L(\mathcal{B})$, then the containment reduces to emptiness of $L(\mathcal{A}) \cap L(\bar{\mathcal{B}})$. Hence, each state P of $\mathcal{T}$ needs to contain additionally information on a configuration of $\mathcal{A}$. Due to the fact that $\mathcal{A}$ is purely-existential, P will contain a unique pair $(q, \mathbf{v})$, where q is a state of $\mathcal{A}$ and $\mathbf{v}$ a valuation of all its clocks. The transition relation $P \xrightarrow{a,t} P'$ is adapted so that the delay t before performing an action a is the same in $\mathcal{A}$ and $\mathcal{B}$. This guarantees that the facts analogous to Lemma 1 and 2 hold; but now a state P is *bad* iff all states of *both* $\mathcal{A}$ and $\mathcal{B}$ appearing in P are accepting.

Definition of H is precisely as before, but it needs a preprocessing: the pair $(q, \mathbf{v})$ corresponding to $\mathcal{A}$ is split into a number of triples $(q, \mathbf{v}_x, x)$, one for each clock x of $\mathcal{A}$. The triples are identical on the first component, and $\mathbf{v}_x$ is the value of clock x. Observe that the number of such pairs is the same in each state of $\mathcal{H}$, and equal to the number of clocks in $\mathcal{A}$. An analog of Lemma 5 holds: $L(\mathcal{A}) \cap L(\bar{\mathcal{B}})$ is nonempty iff a bad state is reachable in $\mathcal{H}$.

Finally, Lemma 7 and 9 hold as well, and the proofs are similar. The proofs of Lemma 8 and 10 rest unchanged.

4 Lower Bound

In this section we prove the following lower bound result.

Theorem 3. *The complexity of the emptiness problem for one-clock purely universal alternating timed automata is not bounded by a primitive recursive function.*

Since emptiness and universality are dual in the setting of alternating automata, as a direct conclusion we get the following:

Corollary 2. *The complexity of the universality problem for one-clock purely existential alternating (i.e., nondeterministic) timed automata is not bounded by a primitive recursive function.*

This answers the question posed by Ouaknine and Worrell [20].

The rest of this section contains the proof of Theorem 3. The proof is a reduction of the reachability problem for *lossy one-channel systems* [22].

Definition 8 (Channel system). *A* channel system *is given by a tuple $\mathcal{S} = (Q, q_0, \Sigma, \Delta)$, where Q is a finite set of control states, $q_0 \in Q$ is an initial state, Σ is a finite channel alphabet and $\Delta \subseteq Q \times (\{!a : a \in \Sigma\} \cup \{?a : a \in \Sigma\} \cup \{\epsilon\}) \times Q$ is a finite set of transition rules.*

A configuration of $\mathcal{S}$ is a pair (q, w) of a control state q and a channel content $w \in \Sigma^*$. Transition rules allow the system to pass from one configuration to another. In particular, a rule $\langle q, !a, q' \rangle$ allows in a state q to write to the channel and to pass to the new state q'. Similarly, $\langle q, ?a, q' \rangle$ means reading from a channel and is allowed in state q only when a is at the end of the channel. The channel is a FIFO, and by convention $\mathcal{S}$ writes at the beginning and reads at the end. Finally, a rule $\langle q, \epsilon, q' \rangle$ allows for a silent change of control state, without reading or writing.

Formally, there is a (perfect) transition $(q, w) \xrightarrow{\gamma} (q', w')$ if one of the following conditions is satisfied:

- $\gamma = \langle q, \epsilon, q' \rangle$ and $w = w'$, or
- $\gamma = \langle q, !a, q' \rangle$ for some $a \in \Sigma$, and $w' = aw$, or
- $\gamma = \langle q, ?a, q' \rangle$ for some $a \in \Sigma$, and $w = w'a$.

The *initial configuration* is (q_0, ϵ), i.e., execution of $\mathcal{S}$ starts with the empty channel. For technical convenience, we assume w.l.o.g. that there is no rule returning back to the initial state: for each rule $\langle q, x, q' \rangle \in \Delta$, $q' \neq q_0$.

A *lossy channel system* differs from the perfect one in only one respect: during the transition step, an arbitrary number of messages stored in the channel may be lost. To define lossy transitions, we need the subsequence ordering on Σ^*, denoted by $\sqsubseteq$ (e.g., `tata` $\sqsubseteq$ `atlanta`). We say that there is a lossy transition from (q, w) to (q', w'), denoted by $(q, w) \stackrel{\gamma}{\Longrightarrow} (q', w')$, iff there exists $u, u' \in \Sigma^*$ such that $u \sqsubseteq w$, $(q, u) \stackrel{\gamma}{\longrightarrow} (q', u')$ and $w' \sqsubseteq u'$.

By a *lossy computation* of a channel system $\mathcal{S}$ we mean a finite sequence:

$$(q_0, \epsilon) \stackrel{\gamma_1}{\Longrightarrow} (q_1, w_1) \stackrel{\gamma_2}{\Longrightarrow} (q_2, w_2) \quad \dots \quad \stackrel{\gamma_n}{\Longrightarrow} (q_n, w_n). \tag{2}$$

Definition 9. Lossy reachability problem *for channel systems is: given a channel system S and a configuration (q_f, w_f), with $q_f \neq q_0$, decide whether there is a lossy computation of S ending in (q_f, w_f).*

Theorem 4 ([22]). *The lossy reachability problem for channel systems has non-primitive recursive complexity.*

The result of [22] was showed for a slightly different model. Namely, during a single transition, a finite sequence of messages was allowed to be read or written to the channel. Clearly, reachability problems in both models are polynomial-time equivalent.

In the sequel we describe a reduction from the lossy reachability for channel systems to the emptiness problem for one-clock purely-universal alternating timed automata. Given a channel system $\mathcal{S} = (Q, q_0, \Sigma, \Delta)$, and a configuration (q_f, w_f), we effectively construct a purely-universal automaton $\mathcal{A}$ with a single clock x, and the input alphabet $\overline{\Sigma} = Q \cup \Sigma \cup \Delta$. The construction will assure that $\mathcal{A}$ accepts precisely correct encodings of lossy computations of $\mathcal{S}$ ending in (q_f, w_f). A computation as in (2) will be encoded as the following word over $\overline{\Sigma}$:

$$q_n \gamma_n w_n \quad q_{n-1} \gamma_{n-1} w_{n-1} \quad \dots \quad q_1 \gamma_1 w_1 \quad q_0, \tag{3}$$

where $q_i \in Q$, $\gamma_i \in \Delta$, $w_i \in \Sigma^*$. Let $\mathcal{S}$ be fixed in this section.

It will be convenient here to write timed words in a slightly different way than before. From now on, whenever we write a word $w = (a_1, t_1)(a_2, t_2) \dots (a_n, t_n)$ we mean that the letter a_i appeared t_i time units after the beginning of the word. In particular, a_{i+1} appeared $t_{i+1} - t_i$ time units after a_i. Clearly this is correct only when $t_{i+1} \geq t_i$, for $i = 1 \dots n-1$.

Before the formal definition of encoding of a computation by a timed word we outline shortly the underlying intuition. We will require that the letter q_n appears at time 0 and then that each letter q_i appears at time $n - i$. Hence, each configuration will be placed in a unit interval. To ensure consistency of the channel contents at consecutive configurations we require that if a message survived during a step i (it was neither read nor written nor lost) then the distance in time between its appearances in the sequences w_i and w_{i-1} should be precisely 1.

We will need a new piece of notation : by $(w + 1)$ we mean the word obtained from w by increasing all t_i by one time unit, i.e., $(w + 1) = (a_1, t_1 + 1)(a_2, t_2 + 1) \dots (a_n, t_n + 1)$.

Definition 10. *By a* lossy computation encoding ending in (q_f, w_f) *we mean any timed word over* $\overline{\Sigma}$ *of the form:*

$$(q_n, t_n)(\gamma_n, t'_n)v_n \ (q_{n-1}, t_{n-1})(\gamma_{n-1}, t'_{n-1})v_{n-1} \quad \ldots \quad (q_1, t_1)(\gamma_1, t'_1)v_1 \ (q_0, t_0),$$

where each $v_i = (a_i^1, u_i^1) \ \ldots \ (a_i^{l_i}, u_i^{l_i})$ *is a timed word over* Σ*. Additionally we require that for each* $i \leq n$ *and* $j = 1, \ldots, l_i$*, the following conditions hold:*

(P1) *Structure:*

$$q_i \in Q, \gamma_i \in \Delta, a_i^j \in \Sigma, \gamma_i = \langle q_{i-1}, x, q_i \rangle, q_n = q_f \text{ and } a_n^1 \ldots a_n^{l_n} = w_f.$$

(P2) *Distribution in time:*

$$n - i = t_i < t'_i < u_i^1 < u_i^2 < \ldots < u_i^{l_i} < t_{i+1} = n - i + 1.$$

(P3a) *Epsilon move: if* $\gamma_i = \langle q_{i-1}, \epsilon, q_i \rangle$ *then* $(v_i + 1) \sqsubseteq v_{i-1}$*.*
(P3b) *Write move: if* $\gamma_i = \langle q_{i-1}, !a, q_i \rangle$ *then either* $v_i = (a, u_i^1)v'$ *and* $v' + 1 \sqsubseteq v_{i-1}$*, or* $(v_i + 1) \sqsubseteq v_{i-1}$*.*
(P3c) *Read move: if* $\gamma_i = \langle q_{i-1}, ?a, q_i \rangle$ *then* $v_{i-1} = v'(a, t)v''$ *for some timed words* v', v'' *and* $t \in \mathbb{R}_+$*, such that* $(v_i + 1) \sqsubseteq v'$*.*

Lemma 11. *$\mathcal{S}$ has a computation of the form (2) ending in* $(q_n, w_n) = (q_f, w_f)$ *if and only if there exists a lossy computation encoding ending in* (q_f, w_f) *as in Definition 10.*

Our aim is:

Lemma 12. *A purely universal automaton $\mathcal{A}$ can be effectively constructed such that $L(\mathcal{A})$ contains precisely all lossy computation encodings ending in* (q_f, w_f)*.*

The proof of this lemma will occupy the rest of this section. Automaton $\mathcal{A}$ will be defined as a conjunction of four automata, each responsible for some of the conditions from Definition 10:

$$\mathcal{A} := \mathcal{A}_{\text{struct}} \ \wedge \ \mathcal{A}_{\text{unit}} \ \wedge \ \mathcal{A}_{\text{strict}} \ \wedge \ \mathcal{A}_{\text{check}}.$$

All four automata will be purely universal and will use at most one clock. Automaton $\mathcal{A}_{\text{struct}}$ verifies condition *(P1)*, automata $\mathcal{A}_{\text{unit}}$ and $\mathcal{A}_{\text{strict}}$ jointly check condition *(P2)*, and $\mathcal{A}_{\text{check}}$ enforces the most involved conditions *(P3a) – (P3c)*.

We omit an obvious definition of $\mathcal{A}_{\text{struct}}$. We also omit the construction of the automaton $\mathcal{A}_{\text{unit}}$ checking that letters from Q appear precisely at times $0, 1, \ldots, n$. Automaton $\mathcal{A}_{\text{strict}}$ will accept a timed word iff the first letter is at time 0 and no two consecutive letters appear at the same time. This can be easily achieved by the following rules:

$$s_0, \overline{\Sigma}, x = 0 \mapsto (s, \emptyset) \qquad s, \overline{\Sigma}, x > 0 \mapsto (s, \{x\}).$$

with s_0 an initial state and both s_0, s as accepting ones. For readability of notation, when no clock is reset, as in the first rule above, we will omit writing it explicitly. Moreover, for conciseness, we implicitly assume that the automaton fails to accept from a state, if no rule is applicable in that state.

The above mentioned automata are not only purely universal but also purely existential, i.e., deterministic. The power of universal choice will be only used in the last automaton $\mathcal{A}_{\text{check}}$, that checks for correctness of each transition step of $\mathcal{S}$. While analysing definition of $\mathcal{A}_{\text{check}}$ we will comfortably assume that an input word meets all conditions verified by the other automata, otherwise the word is anyway not accepted.

The transition rules of $\mathcal{A}_{\text{check}}$ from the initial state s_0 are as follows:

$$\begin{aligned} s_0, q, tt &\mapsto s_0 \wedge (s_{\text{step}}, \{x\}), \qquad \text{for } q \in Q \setminus \{q_0\} \\ s_0, q_0, tt &\mapsto \top \\ s_0, \Sigma \cup \Delta, tt &\mapsto s_0. \end{aligned}$$

Intuitively, at each $q \in Q$, except at q_0, an extra automaton is run from the state s_{step}, in order to check correctness of a single step. Symbol $\top$ on the right-hand side stands for a distinguished state that accepts unconditionally.

Now the rules $s_{\text{step}}, \gamma, \ldots \mapsto \ldots$ depend on $\gamma = \langle q, x, q' \rangle$. There are three cases, corresponding to conditions *(P3a), (P3b)* and *(P3c)*, respectively.

I. Case $\gamma = \langle q, \varepsilon, q' \rangle$*:* $\qquad s_{\text{step}}, \langle q, \varepsilon, q' \rangle, tt \mapsto s_{\text{channel}}.$

In state s_{channel}, the automaton checks the condition (P3a), i.e., whether all consecutive letters from Σ are copied one time unit later. This is done by:

$$\begin{aligned} s_{\text{channel}}, a, tt &\mapsto s_{\text{channel}} \wedge (s_a^{+1}, \{x\}), \quad \text{for } a \in \Sigma \\ s_{\text{channel}}, q, tt &\mapsto \top, \qquad \text{for } q \in Q. \end{aligned}$$

Hence, the automaton starts a check from s_a^{+1} at every letter read. Note that this is precisely here where the universal branching is essential. The task of s_a^{+1} is to check that there is letter a one time unit later:

$$\begin{aligned} s_a^{+1}, a, x = 1 &\mapsto \top \\ s_a^{+1}, \overline{\Sigma}, x < 1 &\mapsto s_a^{+1}. \end{aligned}$$

II. Case $\gamma = \langle q, !a, q' \rangle$*:* $\qquad s_{\text{step}}, \langle q, !a, q' \rangle, tt \mapsto s_{!a}.$

From state $s_{!a}$ the automaton is responsible for checking the correctness of the operation $!a$, i.e., condition (P3b):

$$\begin{aligned} s_{!a}, a, tt &\mapsto s_{\text{channel}} \\ s_{!a}, b, tt &\mapsto (s_b^{+1}, \{x\}) \wedge s_{\text{channel}}, \quad \text{for } b \in \Sigma \setminus \{a\} \\ s_{!a}, q, tt &\mapsto \top, \quad \text{for } q \in Q. \end{aligned}$$

First rule reads simply the letter a and then starts the check from s_{channel}. This is the correct behaviour both when the written message was not forgotten, and when after forgetting it the first message is still a. The second rule deals with the case when the a written to the channel has been lost immediately. The last rule deals with the case when not only the a has been lost, but moreover the channel is empty.

III. Case $\gamma = \langle q, ?a, q' \rangle$*:* $\qquad s_{\text{step}}, \langle q, ?a, q' \rangle, tt \mapsto s_{?a} \wedge (s_{\text{try}?a}, \{x\}).$

The behaviour of $s_{?a}$ is very similar to s_{channel} but additionally it will start a new copy of the automaton in the state $s_{\text{try}?a}$. The goal of $s_{\text{try}?a}$ is to check for the letter a at the end of the present configuration.

$$\begin{aligned} s_{?a}, b, tt &\mapsto s_{?a} \wedge (s_b^{+1}, \{x\}) \wedge (s_{\text{try}?a}, \{x\}), \quad \text{for } b \in \Sigma \\ s_{?a}, Q, tt &\mapsto \top. \end{aligned}$$

Note the clock reset when entering to $s_{\text{try}?a}$. As we cannot know when the configuration ends we start $s_{\text{try}?a}$ at each letter read. If we realize that this was not the end (we see another channel letter) then the check just succeeds. If this was the end (we see a state) then the true check starts from the state $s_{\text{check}?a}$:

$$\begin{aligned} s_{\text{try}?a}, \Sigma, tt &\mapsto \top \\ s_{\text{try}?a}, Q, tt &\mapsto s_{\text{check}?a}. \end{aligned}$$

From $s_{\text{check}?a}$ we look for some a that appears more than one time unit later:

$$\begin{aligned} s_{\text{check}?a}, \overline{\Sigma}, x \leq 1 &\mapsto s_{\text{check}?a} \\ s_{\text{check}?a}, a, x > 1 &\mapsto \top \\ s_{\text{check}?a}, b, x > 1 &\mapsto s_{\text{check}?a}, \quad \text{for } b \in \Sigma \backslash \{a\}. \end{aligned}$$

Automaton $\mathcal{A}_{\text{check}}$ has no other accepting states but $\top$.

By the very construction, $\mathcal{A}$ satisfies Lemma 12. By Lemma 11, $\mathcal{S}$ has a computation (2) ending in (q_f, w_f) if and only if $L(\mathcal{A})$ is nonempty. This completes the proof of Theorem 3.

5 Silent transitions

In this section we point out that the alternating timed automata model cannot be extended with ϵ-transitions. It is known that ϵ-transitions extend the power of nondeterministic timed automata [3, 11]. Here we show some evidence that every extension of alternating timed automata with ϵ-transitions will have undecidable emptiness problem.

It turns out that there are many possible ways of introducing ϵ-transitions to alternating timed automata. To see the issues involved consider the question of whether such an automaton should be allowed to start uncountably many copies of itself or not. Facing these problems we have decided not discuss virtues of different possible definitions but rather to show where the problem is. We will show that the universality problem for purely existential automata with a very simple notion of ϵ-transitions is undecidable.

Timed words are written here in the same convention as in previous section: $w = (a_1, t_1)(a_2, t_2) \dots (a_n, t_n)$ means that the letter a_i appeared at time t_i since the beginning of the computation.

We consider purely existential (i.e. nondeterministic) automata with one clock. We equip them now with additional ϵ-transitions of the form $q, \epsilon, \sigma \;\mapsto\; b$. The following trick is used to shorten formal definitions.

Definition 11. *A* nondeterministic timed automaton with ϵ-transitions *over Σ is a nondeterministic timed automaton over the alphabet $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$.*

For convenience, we want to distinguish an automaton $\mathcal{A}$ with ϵ-transitions over Σ from the corresponding automaton over Σ_ϵ; the latter will be denoted $\mathcal{A}_\epsilon$. Given a timed word v over Σ_ϵ, by $|v|_\epsilon$ we mean the timed word over Σ obtained from w by erasing all (timed) occurrences of ϵ.

Definition 12. *A timed word over Σ is accepted by a timed automaton $\mathcal{A}$ with ϵ-transitions if there is a timed word v over Σ_ϵ accepted by $\mathcal{A}_\epsilon$ such that $w = |v|_\epsilon$.*

Note that according to the definition, an accepting run is always finite. The main result of this section is:

Theorem 5. *The universality problem for one-clock nondeterministic timed automata with ϵ-transitions is undecidable.*

The proof is by reduction of the reachability problem for perfect channel systems, defined similarly as lossy reachability in Definition 9, but w.r.t. *perfect computation* of channel systems. Not surprisingly, a perfect computation is any finite sequence of (perfect) transitions:

$$(q_0, \epsilon) \xrightarrow{\gamma_1} (q_1, w_1) \xrightarrow{\gamma_2} (q_2, w_2) \quad \dots \quad \xrightarrow{\gamma_n} (q_n, w_n),$$

Theorem 6 ([10]). *The perfect reachability problem for channel systems is undecidable, assumed $|\Sigma| \geq 2$.*

Given a channel system $\mathcal{S} = (Q, q_0, \Sigma, \Delta)$ and a configuration (q_f, w_f), we effectively construct a one-clock nondeterministic timed automaton with ϵ-transitions $\mathcal{A}'$ over $\overline{\Sigma}$. Automaton $\mathcal{A}'$ will accept precisely the complement of the set of all *perfect computation encodings ending in* (q_f, w_f), defined by:

Definition 13. *A* perfect computation encoding ending in (q_f, w_f) *is defined as in Definition 10, but with the conditions* (P3a) – (P3c) *replaced by:*

(P3a) *if $\gamma_i = \langle q_{i-1}, \epsilon, q_i \rangle$ then $(v_i + 1) = v_{i-1}$,*
(P3b) *if $\gamma_i = \langle q_{i-1}, !a, q_i \rangle$ then $(v_i + 1) = (a, t)v_{i-1}$, for some $t \in \mathbb{R}_+$.*
(P3c) *if $\gamma_i = \langle q_{i-1}, ?a, q_i \rangle$ then $(v_i(a, t) + 1) = v_{i-1}$, for some $t \in \mathbb{R}_+$.*

Since each perfect computation encoding is a lossy one, $\mathcal{A}'$ will be defined as a disjunction, $\mathcal{A}' \;:=\; \neg\mathcal{A} \;\vee\; \widehat{\mathcal{A}}$, of the complement of the automaton $\mathcal{A}$ from the previous section and another automaton $\widehat{\mathcal{A}}$. As automaton $\neg\mathcal{A}$ takes care of all timed words that are not lossy computation encodings, it is enough to have:

Lemma 13. *Automaton $\widehat{\mathcal{A}}$ accepts precisely these lossy computation encodings ending in (q_f, w_f) that are not perfect computation encodings.*

This will be enough for correctness of our reduction: $\mathcal{A}'$ will accept precisely the complement of the set of all perfect computation encodings.

In the rest of this section we sketch the construction of the automaton required by Lemma 13.

When defining the behaviour of $\widehat{\mathcal{A}}$ we can conveniently assume that the input word is already a lossy computation encoding. The aim of $\widehat{\mathcal{A}}$ is to find a loss of a message in the channel. This will be achieved, roughly, via an ϵ-rule trying to guess a moment t in time such that there is no message occurrence at time t but there is one at time t+1. Of course, $\widehat{\mathcal{A}}$ (and hence $\mathcal{A}'$ as well) will have a single clock x and the input alphabet is $\overline{\Sigma} = Q \cup \Sigma \cup \Delta$.

The transition rules of $\widehat{\mathcal{A}}$ from the initial state s_0 are:

$$\begin{aligned} s_0, q, tt &\mapsto s_0 \vee s_{\text{step}} \quad \text{for } q \in Q \setminus \{q_0\} \\ s_0, \Sigma \cup \Delta, tt &\mapsto s_0. \end{aligned}$$

Intuitively, at each $q \in Q$, except at q_0, $\widehat{\mathcal{A}}$ chooses either to check correctness of this single step or to skip it. $\widehat{\mathcal{A}}$ will have no accepting states but $\top$ that we will use later.

Now the rules $s_{\text{step}}, \gamma, \ldots \mapsto \ldots$ for state s_{step} depend on $\gamma = \langle q, x, q' \rangle$. There are three cases, corresponding to conditions *(P3a), (P3b)* and *(P3c)*, respectively. As the rules follow a similar pattern to that in Section 4, we present only the simplest case when $\gamma = \langle q, \epsilon, q' \rangle$.

$$s_{\text{step}}, \langle q, \epsilon, q' \rangle, tt \mapsto (s_{\text{channel}}, \{x\}).$$

In state s_{channel}, the automaton searches for a message loss. Here we need ϵ-transitions to choose the right moment to move to state s^{+1}:

$$\begin{aligned} s_{\text{channel}}, \epsilon, x > 0 &\mapsto (s^{+1}, \{x\}) \\ s_{\text{channel}}, \Sigma, tt &\mapsto (s_{\text{channel}}, \{x\}) \end{aligned}$$

The task in state s^{+1} is to wait precisely one time unit and then check for a letter, similarly as state s_a^{+1} in Section 4. Transition from s_{channel} to s^{+1} is only possible when $x > 0$. As x is reset at each letter read, this ensures a positive delay between any letter and an ϵ-move.

$$\begin{aligned} s^{+1}, \overline{\Sigma}, 0 < x < 1 &\mapsto s^{+1} \\ s^{+1}, \overline{\Sigma}, x = 1 &\mapsto \top \end{aligned}$$

The only way of accepting from s^{+1} is to consume a number of letters while $0 < x < 1$ and finally find a letter at $x = 1$. Note strictness of the left-hand side inequality in $0 < x < 1$. It is crucial here and excludes $x = 0$, that would mean that some letter occurred in the input word at the moment of the ϵ-move that entered into s^{+1}.

This completes our description of the construction of the automaton $\widehat{\mathcal{A}}$ as required by Lemma 13. Having it we have the automaton $\mathcal{A}'$ which shows Theorem 5.

6 Infinite words

In this section we consider one-clock alternating timed automata over infinite words with Büchi acceptance condition. The acceptance game is defined as in Section 2, but it is played over an infinite word

$$(a_1, t_1)(a_2, t_2)\dots$$

Hence each play $(q_0, \mathbf{v}_0), (q_1, \mathbf{v}_1), \dots$ is infinite. The winner is Eve iff an accepting state appears infinitely often, i.e., $q_i \in F$ for infinitely many indices i. In this section we prove the following result.

Theorem 7. *The universality problem for one-clock nondeterministic Büchi timed automata is undecidable.*

As a direct corollary, emptiness problem of one-clock alternating automata is undecidable as well.

To prove Theorem 7 we code the halting problem. We can assume that the Turing machine starts the empty tape and accepts by reaching a unique accepting state q_{acc}. Furthermore, we assume that the machine is deterministic, i.e., we have a transition function δ specifying for each control state q and tape symbol a a triple $\delta(q, a) = (d, q', b)$ consisting of a head direction $d \in \{\leftarrow, \cdot, \rightarrow\}$, new state q' and letter b to be written onto the tape in place of a.

The idea of the reduction is based on the fact that instead of considering a computation that just stops in an accepting state we will encode existence of a computation that after reaching an accepting state clears the tape with blanks and restarts. Thus the accepting computation is rather a repetitive accepting computation.

We code a sequence of configurations as before, each configuration should fit in a unit interval. We make our simulation in such a way that the first configuration is already of length sufficient for the whole computation, hence in the simulation of machine steps we would never have to add or remove tape positions.

Nondeterministic automaton we are going to construct will accept the sequences that are not encodings of the repetitive accepting computation of the machine. With one clock we can check that there is a cheating, i.e., letter a in one configuration is changed to b in the next although it should have not. We can also check that a letter disappeared (it was in one configuration and not in the next). What we cannot check directly is that there are new letters in the next configuration, i.e., there can appear new tape positions that were not there before. But if this kind of inserts happen infinitely often then we can find a sequence of tape symbols appearing at times $t_1 < t_2 < \dots$ such that the sequence

$\mathtt{fract}(t_1)$, $\mathtt{fract}(t_2)$, ... is either strictly increasing or strictly decreasing. This can be checked by a nondeterministic Büchi automaton. Hence, we can construct an automaton that does not accept the sequences where there are no cheatings, no disappearances and only finitely many inserts. In such a sequences after some position we have a correct and accepting computation of the Turing machine. Thus, the nondeterministic automaton will not accept some word iff the machine halts, i.e., accepts from the empty tape.

Now we will make all these intuitions more formal. Let $\mathcal{M}$ be a fixed Turing machine in the rest of this section; by Q and Σ let us denote the set of control states and tape alphabet of $\mathcal{M}$, respectively. Assume that a blank symbol $\mathtt{B}$ is in Σ. Given $\mathcal{M}$, we will effectively construct a nondeterministic Büchi automaton $\mathcal{A}$ with a single clock x over the input alphabet $\overline{\Sigma} = Q \cup \Sigma \cup \Sigma \times \{\mathtt{H}\}$. A letter $\langle a, \mathtt{H}\rangle$, for $a \in \Sigma$, represents a tape symbol a with the head over it. We put $\Sigma_{\mathtt{H}} = \Sigma \cup \Sigma \times \{\mathtt{H}\}$.

The configuration of $\mathcal{M}$ is a pair (q, w) consisting of a control state $q \in Q$ and a word $w \in {\Sigma_{\mathtt{H}}}^*$ representing the tape content. The transition function δ of $\mathcal{M}$ gives rise to a relation between configurations, describing the single step of $\mathcal{M}$. We will denote this by $qw \longrightarrow q'w'$, to say that a single step from configuration (q, w) yields a new configuration (q', w') and that w and w' are of the same length. So we will model computation that does not go outside w with the idea that enough space was allocated in the initial configuration.

This notation assumes a fixed size of tape available, i.e., w and w' are of the same length and the head may not move outside w. For convenience, we will also write $qv \rightsquigarrow q'v'$ for *timed* words v and v' if $q\, \mathtt{untime}(v) \longrightarrow q'\mathtt{untime}(v')$ holds and time-stamps are identical in v and v' (note that v and v' are of the same length in particular); $\mathtt{untime}(v)$ stands for the word v after removing time-stamps.

Definition 14. *By a* recurrent accepting computation encoding *we mean any timed word w over $\overline{\Sigma}$ of the form:*

$$(q_0, t_0)\ v_0\ (q_1, t_1)\ v_1 \quad \ldots,$$

such that the following conditions hold:

(P1) *Structure: each $q_i \in Q$ and each $v_i = (a_i^1, u_i^1)\ \ldots\ (a_i^{l_i}, u_i^{l_i})$ is a nonempty finite timed word over $\Sigma_{\mathtt{H}}$ such that precisely one of $a_i^1 \ldots a_i^{l_i}$ is in $\Sigma \times \{\mathtt{H}\}$.*

(P2) *Distribution in time: $i = t_i < u_i^1 < u_i^2 < \ldots < u_i^{l_i} < t_{i+1} = i+1$.*

(P3) *Acceptance: q_0 is the initial state of $\mathcal{M}$, each of $a_0^1 \ldots a_0^{l_0}$ is in $\{\mathtt{B}, \langle \mathtt{B}, \mathtt{H}\rangle\}$, and $q_i = q_{acc}$ for infinitely many i.*

(P4) *Recurrence: whenever $q_{i-1} = q_{acc}$, then $q_i = q_0$ and $a_i^1, \ldots, a_i^{l_i} \in \{\mathtt{B}, \langle \mathtt{B}, \mathtt{H}\rangle\}$.*

(P5) *Steps: whenever $q_{i-1} \neq q_{acc}$, $q_{i-1}(v_{i-1} + 1) \rightsquigarrow q_i v$, for some $v \sqsubseteq v_i$.*

(P6) *Insertions bound: w contains no infinite subsequence $(a_0, u_0)(a_1, u_1)\ldots$ such that $u_0 < u_1 < \ldots$, $a_i \in \Sigma_{\mathtt{H}}$ for all $i \geq 0$, and the sequence*

$$\mathtt{fract}(u_0), \mathtt{fract}(u_1), \ldots$$

is either strictly increasing or strictly decreasing.

Lemma 14. *Started with the empty tape, the machine $\mathcal{M}$ accepts if and only if there exists a recurrent accepting computation encoding as in Definition 14.*

Proof. Assume $\mathcal{M}$ accepts. There is a sequence

$$q_0 w_0 \longrightarrow q_1 w_1 \ldots \longrightarrow q_n w_n$$

where $q_n = q_{\mathrm{acc}}$ and w_0 is a finite word over $\Sigma_{\mathtt{H}}$ representing a sufficiently big portion of initially empty tape to store the computation. Hence, there is a recurrent accepting computation encoding obtained by repeating infinitely the word $q_0 w_0 q_1 w_1 \ldots q_n w_n$; time-stamps for tape symbols in w_0, w_1, $\ldots$ can be chosen arbitrarily to satisfy *(P2)* and *(P5)*.

For the opposite direction, assume that some recurrent accepting computation encoding w exists.

By *(P6)*, it contains only finitely many *insertions*, where by an insertion we mean a pair (a, t), $a \in \Sigma_{\mathtt{H}}$, appearing in w such that no letter appears at time $t-1$ in w. Indeed, assume otherwise, i.e., assume that the number of insertions in w is infinite. Build the infinite sequence of all the insertions, in the order they appear in w. The fractional parts $\mathtt{fract}(t)$ of all the time-stamps form an infinite sequence of reals in $(0..1)$, with no number appearing twice. Such a sequence has necessarily a subsequence that is either strictly increasing or strictly decreasing – contradiction with *(P6)*.

By *(P3)* and *(P4)*, w contains infinitely many restarts of the machine. This implies that there is a restart followed by no insertion any more. Hence, from this position on, the encoding simulates the machine faithfully and provides the halting run of the machine. ◻

The undecidability result will follow from the next lemma.

Lemma 15. *A nondeterministic automaton $\mathcal{A}$ can be effectively constructed such that $L(\mathcal{A})$ contains precisely all timed words that are not recurrent accepting computation encodings.*

The automaton $\mathcal{A}$ is a disjunction of six automata, each of them accepting timed words that do not satisfy one of conditions *(P1)–(P6)*, respectively. We omit the automata for (negation of) *(P1)–(P4)* and focus of the other two conditions only. While analysing the definitions we may assume conveniently that the input word satisfies conditions *(P1)–(P4)*.

Automaton for negation of *(P5)*, in its initial state s_0, at each letter $q \in Q$ read, decides nondeterministically either to check this step, or to keep searching for another step to check; in the former case, it guesses a move of the head in this step:

$$\begin{aligned} s_0, q, tt &\mapsto s^q_{\leftarrow} \vee s^q_{\rightarrow} \vee s^q_{\cdot} \vee s_0, \text{ for } q \in Q \\ s_0, \Sigma_{\mathtt{H}}, tt &\mapsto s_0. \end{aligned}$$

To show the idea, we present in detail the transition rules from state $s^q_{\cdot}$ only; but we omit transitions from $s^q_{\leftarrow}$ and $s^q_{\rightarrow}$, as they are conceptually similar. In state $s^q_\cdot$, the automaton needs to check that the next configuration differs from the configuration determined by a single machine step from the current configuration. The automaton can check tape symbols appearing precisely one unit later that some symbol in the current configuration; hence insertions are pretty allowed.

$$\begin{aligned}
s^q_\cdot, a, tt &\mapsto (s^{+1}_a, \{x\}) \vee s^q_\cdot, \text{ for } a \in \Sigma \\
s^q_\cdot, \langle a, \mathtt{H}\rangle, tt &\mapsto (s^{+1}_{\langle b,\mathtt{H}\rangle}, \{x\}) \vee s^{q'}_{\text{cont}}, \text{ if } \delta(q,a) = (\cdot, q', b) \\
s^q_{\text{cont}}, a, tt &\mapsto (s^{+1}_a, \{x\}) \vee s^q_{\text{cont}} \\
s^q_{\text{cont}}, q', tt &\mapsto \top, \text{ if } q' \neq q. q \in Q.
\end{aligned}$$

Observe that the automaton fails to accept from $s^q_\cdot$ if the head move in current configuration is not '$\cdot$', i.e, the automaton's guess has been incorrect. The task from state s^{+1}_a, for $a \in \Sigma_{\mathtt{H}}$, is merely to check that the letter appearing one unit later is *not* equal to a, or that there is no such letter at all:

$$\begin{aligned}
s^{+1}_a, \overline{\Sigma}, x < 1 &\mapsto s^{+1}_a \\
s^{+1}_a, b, x = 1 &\mapsto \top, \text{ if } a \neq b \\
s^{+1}_a, \overline{\Sigma}, x > 1 &\mapsto \top.
\end{aligned}$$

The only accepting state is $\top$.

Now we switch to condition *(P6)*. The task is to recognize a strictly increasing or strictly decreasing subsequence as defined in *(P6)*, hence the automaton is a disjunction $\mathcal{A}_{\text{inc}} \vee \mathcal{A}_{\text{dec}}$. For simplicity of analysis, assume that the input word satisfies all previous conditions *(P1)–(P5)*. In particular, for each letter appearing at time t, say, there is another letter at time $t+1$.

As a preparation, consider the following transition rules, from states s and $\bar{s}$, respectively:

$$\begin{aligned}
s, \Sigma_{\mathtt{H}}, tt &\mapsto s & \qquad \bar{s}, \Sigma_{\mathtt{H}}, x < 1 &\mapsto \bar{s} \\
s, Q, x < 1 &\mapsto \bar{s} & \qquad \bar{s}, \Sigma_{\mathtt{H}}, x = 1 &\mapsto (s, \{x\}) \\
s, Q, x = 1 &\mapsto (s, \{x\}) & \qquad \bar{s}, Q, tt &\mapsto (\bar{s}, \{x\})
\end{aligned}$$

Imagine that the clock x has been reset at some letter $a \in \Sigma_{\mathtt{H}}$ of the input word. Now, starting from state s, the above rules describe scanning of the word in the following cycle: scan all letters in $\Sigma_{\mathtt{H}}$ staying in state s, then on $q \in Q$ change the state to $\bar{s}$; then scan the following letters in $\Sigma_{\mathtt{H}}$ until $x = 1$, i.e., until precisely one time unit elapses since the last clock reset; then reset the clock again and change to state s; and so on. Hence, the whole word is conceptually split into segments determined by the clock resets, and each segment is typically scanned in two "phases": first the s-phase and then the $\bar{s}$-phase. The transition from s to $\bar{s}$ can happen when we see a state from Q; thus only at integer times by property (P2). The only small difference appears when one of phases starts by a clock reset at some letter $q \in Q$; in this case the other phase is degenerate and the bottom-most transition rules for s and $\bar{s}$ apply. In fact this is the case initially, since for the initial state of $\mathcal{A}_{\mathrm{inc}}$ and $\mathcal{A}_{\mathrm{dec}}$ we choose s and $\bar{s}$, respectively.

Having these rules, definition of $\mathcal{A}_{\mathrm{inc}}$ and $\mathcal{A}_{\mathrm{dec}}$ requires only appropriate handling of moments where additional clock resets may be done. In $\mathcal{A}_{\mathrm{inc}}$ the additional clock resets will be enabled only during s-phase, while in $\mathcal{A}_{\mathrm{dec}}$ only in $\bar{s}$-phase.

We will need a third state s' with the following rules:

$$\begin{aligned} s', \Sigma_{\mathtt{H}}, tt &\mapsto s' \\ s', Q, tt &\mapsto \bar{s}, \end{aligned}$$

enabling to mimic the s-phase, but not enabling for any additional clock reset until some $q \in Q$ is observed. State s' will be the only accepting state in both $\mathcal{A}_{\mathrm{inc}}$ and $\mathcal{A}_{\mathrm{dec}}$ and will be intentionally visited at each consecutive letter belonging to a strictly increasing (or decreasing) subsequence. Now, to complete the definition of $\mathcal{A}_{\mathrm{inc}}$, we allow the transition from s to s' by replacing the first rule for s by the following rule:

$$s, \Sigma_{\mathtt{H}}, tt \mapsto s \vee (s', \{x\}).$$

Notice that we do not allow to reset clock more that once in one s-phase (by the first rule for s'). But as we have assumed *(P1)–(P5)*, we know that each letter reappears, perhaps not identically, one unit later. Hence we will not miss a strictly increasing subsequence, but only "postpone" capturing its next element to the next s-phase.

Similarly, to complete the definition of $\mathcal{A}_{\mathrm{dec}}$, we allow the transition from $\bar{s}$ to s' by replacing the first rule for $\bar{s}$ by the following one:

$$\bar{s}, \Sigma_{\mathtt{H}}, x < 1 \mapsto \bar{s} \vee (s', \{x\}).$$

This completes description of automaton $\mathcal{A}$ needed for the proof of Lemma 15 and hence also the proof of Theorem 7.

7 Final Remarks

In this paper we have explored the possibilities opened by the observation that the universality problem for nondeterministic timed automata is decidable [20]. We have extended this result to obtain a class of timed automata that is closed under boolean operations and that has decidable emptiness problem. We have shown that despite being decidable the problem has prohibitively high complexity. We have also considered the extension of the model with epsilon transitions. The undecidability result for this model points out what makes the basic model decidable and what further extensions are not possible. Finally, maybe somewhat surprisingly, we prove that the universality for 1-clock nondeterministic timed automata but over infinite words is undecidable.

We see several topics for further work:

– Adding event-clocks to the model and/or extending from timed words to trees. It seems that in both cases one would still obtain a decidable model.
– Decidability of the universality problem for one-clock co-Büchi automata is still open.
– Finding logical characterisations of the languages accepted by alternating timed automata with one clock. Since we have the closure under boolean operations, we may hope to find one.
– Finding a different syntax that will avoid the prohibitive complexity of the emptiness problem. There may well be another way of presenting alternating timed automata that will give the same expressive power but for which the emptiness test will be easier.

References

1. P. Abdulla, K. Čerāns, B. Jonsson, and Y. Tsay. General decidability theorems for infinite state systems. In *LICS'96*, p. 313–323, 1996.
2. P. A. Abdulla, J. Deneux, J. Ouaknine, and J. Worrell. Decidability and complexity results for timed automata via channel machines. In *ICALP'05*, LNCS 3580, 2005.
3. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
4. R. Alur, M. Bernadsky, and P. Madhusudan. Optimal reachability for weighted timed games. In *ICALP'04*, volume 3124 of LNCS, p. 122–133, 2004.
5. R. Alur, L. Fix, and T. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theoretical Computer Science*, 204:253-273, 1999.
6. R. Alur,T. Henzinger, M.Y. Vardi. Parametric real–time reasoning. In *Proc. 25th ACM Symp. on Theory of Computing*, p. 592–601, 1993.
7. E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *Proc. IFAC Symp. System Structure and Control*, p. 469–474, 1998.
8. P. Bouyer, F. Cassez, E. Fleury, and K. G. Larsen. Optimal strategies in priced timed game automata. In *FSTTCS'04*, LNCS, 2004.
9. P. Bouyer, D. D'Souza, P. Madhusudan, and A. Petit. Timed control with partial observability. In *CAV'03*, volume 2725 of LNCS, p. 180–192, 2003.
10. D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.
11. B. Bérard, V. Diekert, P. Gastin, and A. Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2):145–182, 1998.
12. F. Cassez, T. A. Henzinger, and J.-F. Raskin. A comparison of control problems for timed and hybrid systems. In *Hybrid Systems Computation and Control (HSCC'02)*, volume 2289 of LNCS, p. 134–148, 2002.
13. M. Dickhöfer, T. Wilke. Timed alternating tree automata: the automata-theoretic solution to the TCTL model checking problem. In *ICALP'99*, volume 1644 of LNCS, p. 281-290, 1999.
14. C. Dima. Real-time automata and the Kleene algebra of sets of real numbers. In *STACS'00*, volume 1170 of LNCS, p. 279–289, 2000.
15. A. Finkel and Ph. Schnoebelen. Well structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.
16. G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc.*, 2(7):326–336, 1952.

17. S. Lasota, I. Walukiewicz. Alternating Timed Automata. In *FOSSACS'05*, volume 3441 of LNCS, p. 250–265, 2005.
18. S. La Torre, M. Napoli. Timed Tree Automata with an Application to Temporal Logic. *Acta Informaticae* 38(2):89-116, 2001.
19. F. Laroussinie, N. Markey, and Ph. Schnoebelen. Model checking timed automata with one or two clocks. In *CONCUR'04*, volume 3170 of LNCS, p. 387–401, 2004.
20. J. Ouaknine and J. Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *LICS'04*, p. 54–63, 2004.
21. J. Ouaknine and J. Worrell. On the decidability of metric temporal logic. In *LICS'05*, 2005.
22. Ph. Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Information Processing Letters*, 83(5):251–261, 2002.